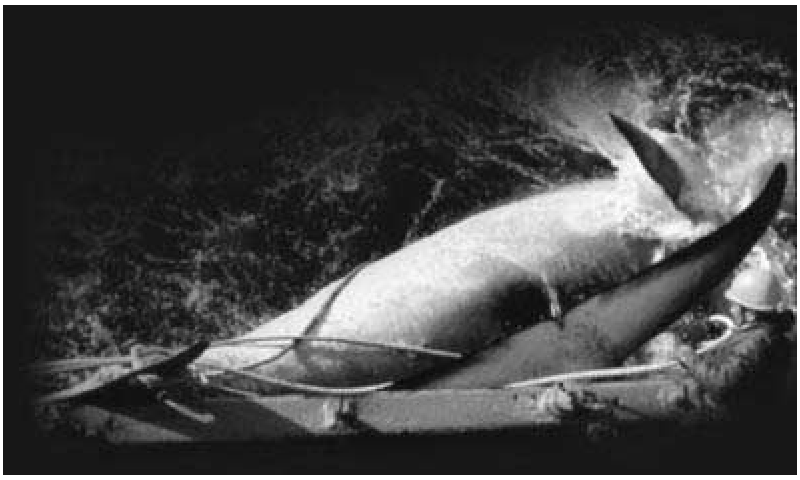
Documentation of a Fortran 77 subroutine implementing the catch limit algorithm



SAMBA/25/00

Ragnar Bang Huseby
Magne Aldrin

November 2000

NR Norsk Regnesentral
ANVENDT DATAFORSKNING

NR-notat/NR Note

Tittel/Title: Documentation of a Fortran 77 subroutine implementing the catch limit algorithm

Dato/Date: November
År/Year: 2000
Notat nr:
Note no: SAMBA/25/00

Forfatter/Author: Ragnar Bang Huseby, Magne Aldrin

Sammendrag/Abstract: This note contains documentation of a Fortran 77 subroutine implementing the catch limit algorithm.

Emneord/Keywords: Stock management, whaling

Tilgjengelighet/Availability: Open

Prosjektnr./Project no.: 63002

Satsningsfelt/Research field: Statistics for environment and marine resources

Antall sider/No. of pages: 94

Norsk Regnesentral / Norwegian Computing Centre
Gaustadalléen 23, Postboks 114 Blindern, 0314 Oslo, Norway
Telefon 22 85 25 00, telefax 22 69 76 60

1 Introduction

The Scientific Committee of The International Whaling Commission has tested various procedures on simulated population and catch histories. In 1991 the Committee chose one procedure, proposed by Cooke, as the core element of the so called "Revised Management Procedure".

This procedure, as specified in [1], has been implemented by the Norwegian Computing Center. The program, called `rmp`, was described in [2]. From this program, the module implementing the catch limit algorithm has been extracted. The module was modified in June 1999, and further changes of the code have been made in June 2000 and November 2000. The version of November 2000 will be described in this note.

The catch limit algorithm is reviewed in Section 2. In Section 3, we describe how the catch limit is computed in our implementation, and in Section 4, we review the numerical analysis methods used.

Appendix A contains a manual description of the subroutine computing the catch limit. Appendix B contains a list of the subroutines of the module. The difference between the various versions of the module is described in Appendix C, and the Fortran code of the most recent version of the module is listed in Appendix D.

2 Catch limit algorithm

In this section, the catch limit algorithm is reviewed. We use the same notation as in [1]. The input data consists of the time series of historic annual catches and the time series of absolute abundance estimates along with the information matrix of the logarithm of the estimates. In our implementation we assume that

1. the abundance estimates are positive, and
2. the information matrix of the logarithm of the estimates is nonnegative definite.

The internal population model of the catch limit algorithm is defined by the following dynamics

$$\begin{aligned} P_0 &= \frac{P_T}{D_T}, \\ P_{t+1} &= P_t - C_t + 1.4184\,\mu\, P_t(1-(\frac{P_t}{P_0})^2) \qquad (0 \le t < T), \end{aligned} \tag{1}$$

where

⋆ P_t is the population size in numbers at the beginning of year t

⋆ C_t is the catch in numbers in year t

⋆ $D_T = P_T/P_0$ is the ratio of the population size at the beginning of year T

to the population size at the beginning of year zero, denoted stock depletion
⋆ Year zero is the first year of the historic catch series used in assessments
⋆ Year T is the year the catch limit is to be applied (i.e. the first year of an assessment cycle). This is assumed to be the year immediately following the last year of historic catch series used in the assessments
⋆ μ is a parameter describing the productivity.

In this model, μ and D_T are regarded as fixed, but unknown parameters, which together determine the population history, as long as there has been any catches. (In the case of no previous catches, a nominal catch of one whale in year 0 is assumed.)

The abundance estimates are assumed to be log-normally distributed with a given information matrix for the log estimates, estimated from the survey data. The formula for the data likelihood is

$$\text{Likelihood}(\mu, D_T, b) \propto \exp\left(-1/2(\mathbf{a} - \mathbf{p} - \beta\mathbf{1})' H(\mathbf{a} - \mathbf{p} - \beta\mathbf{1})\right) \tag{2}$$

where
⋆ $\mathbf{a}$ is the vector of logarithms of the estimates of population size by year;
⋆ $\mathbf{p}$ is the vector of logarithms of the modeled annual population sizes for the years with population estimates, $p_t = \ln(P_t)$;
⋆ β is the logarithm of the bias parameter, thus $b = \exp(\beta)$;
⋆ H is the information matrix of the $\mathbf{a}$ vector. If H is nonsingular, $H = V^{-1}$ where V is (an estimate of) the covariance matrix of the vector $\mathbf{a}$.

The parameters μ, D_T, and b are assigned a prior distribution which is uniform over the region

$$[\mu_{\min}, \mu_{\max}] \times [D_{T,\min}, D_{T,\max}] \times [b_{\min}, b_{\max}], \tag{3}$$

where $\mu_{\min}$, $\mu_{\max}$, $D_{T,\min}$, $D_{T,\max}$, $b_{\min}$, and $b_{\max}$ are constants. Typical values are $\mu_{\min} = 0.0$, $\mu_{\max} = 0.05$, $D_{T,\min} = 0.0$, $D_{T,\max} = 1.0$, $b_{\min} = 0.0$, and $b_{\max} = 1.6667$.
The joint likelihood function of the parameters μ, D_T, and b is now determined. It is given as follows:

$$\text{Posterior}(\mu, D_T, b) \propto \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s, \quad s = 1/16 \tag{4}$$

The presence of a deflation parameter $0 < s < 1$ down-weights the survey information relative to a strict Bayesian approach.

The internal catch limit is the following function of μ, D_T, and P_T:

$$L_T = \begin{cases} 0 & \text{if } D_T \leq IPL \\ 3\mu(D_T - IPL)P_T & \text{if } D_T > IPL \end{cases} \tag{5}$$

where the internal protection level IPL is a control parameter. A typical value of IPL is 0.54. The internal catch limit can be regarded as the catch limit in the hypothetical case of perfect knowledge of population parameters and size. However, in the Bayesian formalism, it is regarded as a random variable, with marginal posterior distribution obtained from the joint posterior distribution of (μ, D_T, b). The actual catch limit z is defined as a certain percentile of the marginal distribution of L_T. Hence z satisfies

$$P(L_T < z|data) \leq \alpha \leq P(L_T \leq z|data) \tag{6}$$

for a given α. A typical value of α is 0.4102.

3 Computation details

Change of variables: Computation of the catch limit involves integration of the right-hand side of (4) over various subsets of the parameter space. In order to avoid solving for the population history for each functional evaluation, the calculation is based on a change of variables from (μ, D_T, b) to (μ, p_0, b) where $p_0 = \ln(P_0)$. The Jacobi determinant, $J(\mu, p_0, b)$, of the mapping from (μ, p_0, b) to (μ, D_T, b) is defined by

$$J(\mu, p_0, b) = \begin{vmatrix} \frac{\partial\mu}{\partial\mu} & \frac{\partial\mu}{\partial p_0} & \frac{\partial\mu}{\partial b} \\ \frac{\partial D_T}{\partial\mu} & \frac{\partial D_T}{\partial p_0} & \frac{\partial D_T}{\partial b} \\ \frac{\partial b}{\partial\mu} & \frac{\partial b}{\partial p_0} & \frac{\partial b}{\partial b} \end{vmatrix} \tag{7}$$

where $|A|$ means the determinant of the matrix A. It follows that

$$J(\mu, p_0, b) = \frac{\partial P_T}{\partial P_0} - D_T. \tag{8}$$

In order to compute $J(\mu, p_0, b)$ we use the recursion

$$\begin{aligned} \frac{\partial P_0}{\partial P_0} &= 1, \\ \frac{\partial P_{t+1}}{\partial P_0} &= (1 + R - 3R(\frac{P_t}{P_0})^2)\frac{\partial P_t}{\partial P_0} + 2R(\frac{P_t}{P_0})^3 \qquad (0 \leq t < T), \end{aligned} \tag{9}$$

where $R = 1.4184\,\mu$.
It is implicitly assumed that p_0 is a monotone function of μ when D_T is fixed. This will be the case if $J(\mu, p_0, b) > 0$ everywhere except possibly on the boundary, or in the limit as $P_0 \to \infty$. This has not been proved in the strict sense. It has, however, always turned out to be the case in our numerical computations. Thus, there is sufficiently strong numerical evidence to regard the question as settled for all practical purposes.

Splitting the integral over the parameter space: We need to find the integral of the right-hand side of (4) over the region defined by (3). This integral is given by

$$\int_{\mu_{\min}}^{\mu_{\max}} \int_{D_{T,\min}}^{D_{T,\max}} \int_{b_{\min}}^{b_{\max}} \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s db\, dD_T\, d\mu. \quad (10)$$

This integral is also equal to

$$\int_{\mu_{\min}}^{\mu_{\max}} \int_{-\infty}^{\infty} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \quad (11)$$

where

$$f(\mu, p_0, b) = \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s \cdot |J(\mu, p_0, b)|, \quad (12)$$

and $|J(\mu, p_0, b)|$ is the absolute value of the Jacobi determinant. Note that D_T is a function of (μ, p_0). In the computation, it is convenient to split the integral at $D_T = IPL$. Thus, the integral is equal to $I_{lower} + I_{upper}$, where

$$I_{lower} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{-\infty}^{p_{0,split}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \quad (13)$$

and

$$I_{upper} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,split}(\mu)}^{\infty} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \quad (14)$$

where $p_{0,split}(\mu)$ is the value of p_0 such that

$$D_T = IPL \quad (15)$$

for a given μ, and IPL is as in (5).
I_{lower} is split further by splitting the range $(-\infty, p_{0,split}(\mu)]$ into the two intervals $(-\infty, p_{0,lowmid}(\mu)]$ and $[p_{0,lowmid}(\mu), p_{0,split}(\mu)]$, where $p_{0,lowmid}(\mu)$ is the value of p_0 such that

$$D_T = \frac{4}{5} D_{T,\min} + \frac{1}{5} IPL \quad (16)$$

for a given μ. By a change of variable from p_0 to u where

$$p_0 = p_{0,lowmid}(\mu) + (p_{0,lowmid}(\mu) - p_{0,split}(\mu))(\frac{2}{1-u} - 1), \quad (17)$$

and

$$\frac{dp_0}{du} = (p_{0,lowmid}(\mu) - p_{0,split}(\mu)) \frac{2}{(1-u)^2}, \quad (18)$$

the integral over $(-\infty, p_{0,lowmid}(\mu)]$ is transformed to an integral over the finite interval $[-1,1]$. Thus $I_{lower} = I^-_{lower} + I^+_{lower}$, where

$$I^-_{lower} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{-1}^{1} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) \frac{dp_0}{du} db\, du\, d\mu, \tag{19}$$

and

$$I^+_{lower} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,lowmid}(\mu)}^{p_{0,split}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu. \tag{20}$$

Similarly, I_{upper} can be written $I_{upper} = I^-_{upper} + I^+_{upper}$, where

$$I^-_{upper} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,split}(\mu)}^{p_{0,highmid}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{21}$$

and

$$I^+_{upper} = \int_{\mu_{\min}}^{\mu_{\max}} \int_{-1}^{1} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) \frac{dp_0}{dv} db\, dv\, d\mu, \tag{22}$$

where $p_{0,highmid}(\mu)$ is the value of p_0 such that

$$D_T = \frac{4}{5} D_{T,\max} + \frac{1}{5} IPL \tag{23}$$

for a given μ,

$$p_0 = p_{0,highmid}(\mu) + (p_{0,highmid}(\mu) - p_{0,split}(\mu))(\frac{2}{1-v} - 1), \tag{24}$$

and

$$\frac{dp_0}{dv} = (p_{0,highmid}(\mu) - p_{0,split}(\mu)) \frac{2}{(1-v)^2}. \tag{25}$$

Setting up an equation for the catch limit: In order to find z such that (6) is satisfied, we need to compute $P(L_T \leq z|data)$ for various values of z. $P(L_T \leq z|data)$ is equal to

$$\frac{\int_{\mu_{\min}}^{\mu_{\max}} \int_{D_{T,\min}}^{D_{T,z}(\mu)} \int_{b_{\min}}^{b_{\max}} \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s db\, dD_T\, d\mu}{\int_{\mu_{\min}}^{\mu_{\max}} \int_{D_{T,\min}}^{D_{T,\max}} \int_{b_{\min}}^{b_{\max}} \text{Prior}(\mu, D_T, b) \cdot \text{Likelihood}(\mu, D_T, b)^s db\, dD_T\, d\mu}, \tag{26}$$

where $D_{T,z}(\mu)$ is the value of D_T such that

$$L_T = z \tag{27}$$

for a given μ. L_T is the internal catch limit defined by (5). The denominator of (26) is equal to $I_{lower} + I_{upper}$. If $z = 0$, the numerator is equal to I_{lower}. If $z > 0$, the numerator is equal to $I_{lower} + I_z$, where I_z is given by

$$I_z = \int_{\mu_{\min}}^{\mu_{\max}} \int_{p_{0,split}(\mu)}^{p_{0,z}(\mu)} \int_{b_{\min}}^{b_{\max}} f(\mu, p_0, b) db\, dp_0\, d\mu, \tag{28}$$

$p_{0,split}(\mu)$ is defined by (15), and $p_{0,z}(\mu)$ is the value of p_0 such that (27) is satisfied for a given μ. It follows that $z = 0$ is the solution of (6) if $I_{lower}/(I_{lower} + I_{upper}) \geq \alpha$. Otherwise, z satisfies

$$\frac{I_{lower} + I_z}{I_{lower} + I_{upper}} = \alpha. \tag{29}$$

Approximating the catch limit: We are now ready to describe a procedure that computes an approximation of the catch limit. In this procedure, the integrals I^-_{lower}, I^+_{lower}, I^-_{upper}, I^+_{upper}, and I_z defined by (19), (20), (21), (22), and (28), respectively, are calculated by numerical integration. The integrals are evaluated as iterated integrals, and the order of integration is as indicated in the equations above. Each iterated integral is approximated by an n-point Gauss-Legendre integration rule, [3]. The integer n, which is kept fixed in this procedure, is the number of functional evaluations in the approximation. Thus, the approximation can be written as a sum

$$\sum_{i=1}^{n} w_i\, g(x_i), \tag{30}$$

where the w_i's are weights, the x_i's are abscissas, and g is the integrand. The weights and the abscissas depend only on the interval of integration and not on the function to be integrated. For a review of the Gauss-Legendre integration rules, see Section 4.1.
The approximation procedure can be divided into the following steps.

1. Calculate the weights and the abscissas in the Gauss-Legendre integration rule approximating the b-integral.

2. Calculate the weights and the abscissas in the Gauss-Legendre integration rule approximating the μ-integral.

3. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,split}(\mu)$ defined by (15). This equation is solved numerically by Brent's method, [4]. For a brief review of Brent's method, see Section 4.2. In order to find the solution, D_T is evaluated for various values of p_0 by using (1). It is assumed that $-5 \leq p_{0,split}(\mu) \leq 50$.

4. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,lowmid}(\mu)$ defined by (16). This equation is solved in the same way as in Step 3. It is assumed that $-5 \leq p_{0,lowmid}(\mu) \leq 50$.

5. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral: calculate the weights and the abscissas in the Gauss-Legendre integration rules approximating the v-integral in (20) and the u-integral in (19). Each weight in the u-integral is multiplied by $\frac{dp_0}{du}$ evaluated at the corresponding abscissa. $\frac{dp_0}{du}$ is given by (18).

6. Evaluate an approximation of $I_{lower} = I^{-}_{lower} + I^{+}_{lower}$. Each integral on the right-hand side is approximated by a triple sum. In order to find the sums the function f defined by (12) is evaluated at various points. At the points satisfying $-5 \leq p_0 \leq 50$, the population history, see (1), the Jacobi determinant, see (8), and the right-hand side of (2), are calculated. Concerning the calculation of the population history, there are some exceptions that occur if P_0 is large or the population size becomes small, see the documentation of the subroutine `pforw` in Appendix B. The right-hand side of (2) can be written as

$$\exp(-\frac{1}{2}(D_3 - \beta\, D_2 + \beta^2 D_1)) \tag{31}$$

where

$$D_1 = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{i,j}, \tag{32}$$

$$D_2 = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{i,j}\,(a_{y_i} - p_{y_i}), \tag{33}$$

$$D_3 = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{i,j}\,(a_{y_i} - p_{y_i})\,(a_{y_j} - p_{y_j}), \tag{34}$$

a_{y_i} and p_{y_i} are the logarithms of the abundance estimate and the modeled population size, respectively, by year y_i, and $H_{i,j}; i = 1, 2, \ldots, n; j = 1, 2, \ldots, n$ are the entries of the information matrix H. The sums D_1, D_2, and D_3 are computed only once for each (μ, p_0). At the points where either $p_0 < -5$ or $p_0 > 50$, $f(\mu, p_0, b)$ is set to zero.

7. For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,highmid}(\mu)$ defined by (23). This equation is solved in the same way as in Step 3. It is assumed that $-5 \leq p_{0,highmid}(\mu) \leq 50$.

8. Calculate the weights and the abscissas relevant for the computation of I^-_{upper} and I^+_{upper}. This is done in the similar way as in Step 5. Gauss-Legendre integration rules approximates the p_0-integral in (21) and the v-integral in (22).

9. Evaluate an approximation of $I_{upper} = I^-_{upper} + I^+_{upper}$. This is similar to Step 6.

10. If $I_{lower}/(I_{lower} + I_{upper}) \geq \alpha$, the catch limit approximation is zero. Otherwise, the catch limit approximation is the solution of (29) found by Brent's method.

    It is assumed that the solution is in $[0, \mu_{\max} A_*]$, where A_* is either A_τ, the most recent abundance estimate, or $A_{\tau-1}$, the second most recent abundance estimate. If $A_\tau < A_{\tau-1}$ and the variance of the second most recent abundance estimate is smaller than the variance of the most recent abundance estimate, $A_* = A_{\tau-1}$. Otherwise, $A_* = A_\tau$.

    In order to compute the left-hand side of (29), the following tasks must be completed.

    (a) For each abscissa in the Gauss-Legendre integration rule approximating the μ-integral, find $p_{0,z}(\mu)$ defined by (27). This equation is solved in the same way as in Step 3.

    (b) Calculate the weights and the abscissas in the Gauss-Legendre integration rules approximating the p_0-integral in (28).

    (c) Evaluate an approximation of I_z. This is similar to Step 6.

    In extreme cases when $f(\mu, p_0, b) \approx 0$ except on a very small subset of the region of integration, the computed approximation of $I_{lower} + I_{upper}$ might be zero. In that case, the procedure fails to compute an approximation of the catch limit. This type of failure becomes less likely as n grows.

Computing the catch limit by an iterative algorithm: The procedure above approximating the catch limit using n-point Gauss-Legendre integration rules is carried out for $n = 8, 16, 32, 64, 128, 256, 512, 1024$, or until the difference between two successive approximations becomes less than a tolerance specifying the required accuracy.

Error handling: Our implementation does not handle the most extreme cases. If there is evidence that the catch limit cannot be computed to the required accuracy, this will be reported through the output of the main routine of the module. For further details, see the specifications of the parameter `IFAIL` in Appendix A.

4 Description of numerical analysis methods

4.1 Gauss-Legendre integration rules

In this section, we give a brief review of the Gauss-Legendre integration rules. For more details, see e.g. Section 2.7 in [3].

A Gauss-Legendre integration rule is a way of approximating the integral of a function over an interval. The approximation is of the form

$$\int_a^b g(x)dx \approx \sum_{i=1}^{n} w_i\, g(x_i). \tag{35}$$

The weights w_i's and the abscissas x_i's are chosen such that

$$\int_a^b q(x)dx = \sum_{i=1}^{n} w_i\, q(x_i). \tag{36}$$

whenever q is a polynomial of degree $\leq 2n-1$. This is the basic idea of Gauss-Legendre integration rules.
The x_i's are the zeros of the polynomial p_n^*, where the polynomials $p_0^*, p_1^*, \ldots$ satisfy the following conditions.

1. p_n^* is a polynomial of degree n.
2. $\int_a^b (p_n^*(x))^2 dx = 1$.
3. $\int_a^b p_m^*(x) p_n^*(x) dx = 0$ whenever $m \neq n$.

The w_i's are given by

$$w_i = -\frac{k_{n+1}}{k_n} \frac{1}{p_{n+1}^*(x_i) p_n^{*\prime}(x_i)} \tag{37}$$

where k_n is the coefficient of x^n in $p_n^*(x)$.
The subroutine GRULE at page 369 in [3] is used in our implementation. This subroutine computes the $m = [(n+1)/2]$ nonnegative abscissas x_i's and the corresponding weights w_i's of the n-point Gauss-Legendre integration rule when the interval of integration is $[-1, 1]$.

In order to find the abscissas x_i''s and the weights w_i''s in the general case when the interval of integration is $[a, b]$, we use the fact that the abscissas are located symmetrically in the interval $[a, b]$ and the weights corresponding to symmetric points are equal. Then the following relations are valid for $i = 1, \ldots, m$:

$$\begin{aligned} x_i' &= c - d\, x_i, \\ x_{m+i}' &= c + d\, x_{m-i+1}, \\ w_i' &= d\, w_i, \\ w_{m+i}' &= d\, w_{m-i+1}, \end{aligned} \tag{38}$$

where $c = (a+b)/2$ and $d = (b-a)/2$.

4.2 Brent's method for solving equations

In this section, we consider the problem of finding the value of x such that $g(x) = c$ where g is a function of one variable. This problem is equivalent to the problem of finding x such that $f(x) = 0$, where $f(x) = g(x) - c$. Brent's method solves the latter problem numerically. The method combines root bracketing, bisection, and inverse quadratic interpolation, [4]. In our implementation we use the function zbrent in [4] with a slight modification. In order to reduce the amount of computation we first search for a solution in a narrow interval. If we do not succeed, we search for a solution in a broader interval. In the implementation of [4] there is no possibility of extending the search interval.

References

[1] Rep. Int. Whal. Commn. 44, 1994, Annex H.

[2] Fenstad,A.M., Helgeland,J., Aldrin,M., Volden,R.: "High accuracy computer program for the IWC catch limit algorithm", SC/45/Mg3, IWC SC Annual Meeting, Kyoto 1993

[3] Davis,P.J. and Rabinowitz, P.: "Methods of numerical integration", Academic Press, 1975.

[4] Press,W.H., Teukolsky,S.A., Vetterling,W.T., Flannery,B.P.: "Numerical Recipes in FORTRAN", Second Edition, Cambridge University Press, 1992.

[5] Smith, B.T., Boyle, J.M., Dongarra, J.J., Garbow, B.S., Ikebe, Y., Klema, V.C., and Moler C.B.: "Matrix Eigensystem Routines", EISPACK Guide Lecture Notes in Computer Science, Vol. 6, Springer-Verlag, 1976.

[6] Martin, Reinsch, and Wilkinson: "Num. Math. 11", pp 181-195, 1968.

[7] Reinsch: "Comm. ACM 16", p 689, 1973.

A CATCHLIMIT - Manual description of the subroutine

Purpose: Calculate the catch limit for a single area according to the algorithm of Section 2.

Restrictions: We assume that the abundance estimates are positive, and the information matrix of the logarithm of the estimates is nonnegative definite. The subroutine is unable to compute the catch limit in extreme cases.

Files: The file `xrmpSub.f` contains the module implementing the catch limit algorithm including the subroutine `CATCHLIMIT`. The file `xrmpSub_inc.f` contains definitions of some common blocks used by the module and must be included.

Specification:

```
 SUBROUTINE CATCHLIMIT(NUM,ABDIM,CATCH,ABEST,INFOMATRX,AB_YEARS,
*      IN_PPROB,IN_MU_MIN,IN_MU_MAX,IN_DT_MIN,IN_DT_MAX,
*      IN_B_MIN,IN_B_MAX,IN_PLEVEL,
*      OUT_QUOTA,accQuota,outDiff,npRule,
*      POP,DEVPOP,
*      IN_INFOLEVEL,IN_IOUT,IFAIL)
```

Parameters:

1. `NUM - integer` Input
   On entry: The length of the catch history. Actual length of `CATCH` array

2. `ABDIM - integer` Input
   On entry: Number of years with nonzero abundance estimates for the area in question.

3. `CATCH(NUM) - real array` Input
   On entry: `CATCH(Y)` is the historic catch in year `Y`; `Y = 1,2,...,NUM`.
   Corresponds to C_t in (1).
   If there has been any catch, `Y = 1` corresponds to the first year of catch ($t = 0$ in (1)).
   `Y = NUM` corresponds to last premanagement year ($t = T - 1$ in (1)).
   If there has been no catch, `NUM` should be equal to 1.
   If `NUM=1` and `CATCH(1)=0`, `CATCH(1)` is set to 1.
   Constraint: `CATCH(1) > 0 if NUM > 1`.

4. `ABEST(ABDIM) - real array` Input
On entry: `ABEST(I)` is the absolute abundance estimates in year `AB_YEARS(I); I = 1,2,...,ABDIM.`
The vector of the logarithms of the entries in this array corresponds to $\mathbf{a}$ in (2).
Constraint: `ABEST(I) > 0`.

5. `INFOMATRX(ABDIM*(ABDIM+1)/2) - real array` Input
On entry: `INFOMATRX(I)` contains the lower triangle of the matrix H in (2) stored row-wise.
Constraint: H is symmetric and nonnegative definite.

6. `R8WORKSPACE(ABDIM*(ABDIM+7)/2) - real*8 array` Workspace
Workspace needed to determine whether the matrix stored in `INFOMATRX` is nonnegative definite.

7. `AB_YEARS(ABDIM) - integer array` Input
On entry: `AB_YEARS(I)` is the year of the absolute abundance estimate `ABEST(I)`. If `AB_YEARS(I) < 1`, the corresponding abundance estimate is treated as if the sighting was performed in year 1.
Constrains: `AB_YEARS(I) < NUM+1`,
`AB_YEARS(1) < AB_YEARS(2) < ... < AB_YEARS(ABDIM)`.

8. `IN_PPROB - real` Input
On entry: Probability level for distribution of L_T.
Corresponds to α in (6).
Typical value: 0.4102

9. `IN_MU_MIN - real` Input
On entry: Minimum value of productivity parameter.
Corresponds to $\mu_{\min}$ in (3).
Typical value: 0.0
Constraint: `IN_MU_MIN` is nonnegative.

10. `IN_MU_MAX - real` Input
On entry: Maximum value of productivity parameter.
Corresponds to $\mu_{\max}$ in (3).
Typical value: 0.05
Constraints: `IN_MU_MAX` is not less than 10^{-20} and `IN_MU_MAX` is not less than `IN_MU_MIN`.

11. `IN_DT_MIN - real` Input
On entry: Minimum value for stock depletion.
Corresponds to $D_{T,\min}$ in (3).
Typical value: 0.0
Constraint: `IN_DT_MIN` is nonnegative.

12. `IN_DT_MAX - real` Input
On entry: Maximum value for stock depletion.
Corresponds to $D_{T,\max}$ in (3).
Typical value: 1.0
Constraint: `IN_DT_MAX` is not less than `IN_DT_MIN`.

13. `IN_B_MIN - real` Input
On entry: Minimum bias.
Corresponds to $b_{\min}$ in (3).
Typical value: 0.0
Constraint: `IN_B_MIN` is nonnegative.

14. `IN_B_MAX - real` Input
On entry: Maximum bias.
Corresponds to $b_{\max}$ in (3).
Typical value: 1.6667
Constraints: `IN_B_MAX` is not less than 10^{-20} and
`IN_B_MAX` is not less than `IN_B_MIN`.

15. `IN_PLEVEL - real` Input
On entry: Internal protection level.
Corresponds to IPL in (5).
Typical value: 0.54
Constraint: $D_{T,\min} \geq IPL \geq D_{T,\max}$.

16. `OUT_QUOTA - real` Output
On exit: Calculated catch limit.

17. `accQuota - real*8` Input
On entry: Tolerance specifying the required accuracy. The iterative algorithm terminates if the difference between two successive approximations of the catch limit (determined by $\frac{n}{2}$-point and n-point Gauss-Legendre integration rules, respectively) is less or equal to `accQuota`. The approximate solution of (29) is determined such that its accuracy is $0.25 \cdot$ `accQuota`.
Typical value: 0.2

18. `outDiff - real*8` Output
On exit: Achieved accuracy, that is the difference between the last two approximations of the catch limit.

19. `npRule - integer` Output
On exit: The number of points used in the numerical integration in the last iteration.

20. `POP(0:NUM+1) - real*8 array` Workspace
Various population size trajectories. Corresponds to P_t in (1).

21. DEVPOP(ABDIM) - real*8 array Workspace
Difference between abundance estimate and population size at years with abundance estimates for various trajectories.

22. IN_INFOLEVEL - integer Input
On entry: Parameter controlling the level of intermediate printout produced by this module. The larger value, the more printout.
Typical values: 0 - no printout,
1 - possible warnings,
2 - as 1 + print each catch limit approximation,
3 - as 2 + print value of integrals,
4 - as 3 + print some integration limits,
5 - as 4 + print input arrays,
6 - as 5 + print D_1, D_2, and D_3 in (32-34),
7 - as 6 + print likelihood and density values.

23. IN_IOUT - integer Input
On entry: Unit determining file for intermediate printout.

24. IFAIL - integer Input/Output
On entry: If the user sets IFAIL to 0 before calling the routine, execution of the program will terminate if the routine detects an error. Before the program is stopped, an error message is output. If the user sets IFAIL to -1 or 1 before calling the routine, the control is returned to the calling program if the routine detects an error. If IFAIL = -1, an error message is output before the control is returned.
On exit: If IFAIL = 0, no error is detected.
If IFAIL = 2, NUM < 1.
If IFAIL = 3, ABDIM < 1.
If IFAIL = 4, NUM > 1 and CATCH(1) is not positive.
If IFAIL = 5, ABEST(I) is not positive for some I.
If IFAIL = 6, the information matrix of the logarithm of the abundance estimates is not nonnegative definite (At least one of the eigenvalues is negative). Due to numerical inaccuracy a singular matrix may be declared as not being nonnegative definite. In such cases, however, the magnitude of the lowest eigenvalue computed by the module is small. This eigenvalue is printed if IN_INFOLEVEL is positive.
If IFAIL = 7, AB_YEARS(I) > NUM for some I, or the sequence AB_YEARS(I); I=1,...,ABDIM; is not strictly increasing.
If IFAIL = 8, IN_PPROB < 0 or IN_PPROB > 1.
If IFAIL = 9, MU_MIN > MU_MAX, MU_MIN < 0, or MU_MAX < 10^{-20}.
If IFAIL = 10, DT_MIN > DT_MAX or DT_MIN < 0.
If IFAIL = 11, B_MIN > B_MAX, B_MIN < 0, or B_MAX < 10^{-20}.
If IFAIL = 12, IN_PLEVEL < DT_MIN or IN_PLEVEL > DT_MAX.
If IFAIL = 13, accQuota is not positive.

If `IFAIL = 14`, `nmax` in include file is less than the number of rule points.
If `IFAIL = 15`, possible inaccuracies in computed population size history.
If `IFAIL = 16`, P_T becomes larger than $0.5 \cdot 10^{30}$.
If `IFAIL = 17`, the Jacobi determinant $J(\mu, p_0, b)$ becomes negative at some point.
If `IFAIL = 18`, for some μ it was not possible to find $p_{0,split}(\mu)$ defined by (15).
If `IFAIL = 19`, for some μ it was not possible to find either $p_{0,lowmid}(\mu)$ defined by (16) or $p_{0,highmid}(\mu)$ defined by (23).
If `IFAIL = 20`, for some μ it was not possible to find the integration interval of the p_0-integral.
If `IFAIL = 21`, the value of z in (28) becomes negative.
If `IFAIL = 22`, the catch limit could not be computed because the computed approximation of (10) is zero.
If `IFAIL = 23`, it was not possible to solve the equation for the catch limit.
If `IFAIL = 24`, the required accuracy was not reached.
If `IFAIL = -2`, the input value of `IFAIL` is illegal. It is assumed that `IFAIL` value should be 0.

B List of subroutines

The module contains the following subroutines and functions.

1. `SUBROUTINE CATCHLIMIT` - Main subroutine and gateway to the module. Performs some tests on input parameters. Calls `checkdat` and `calc_quota`.

2. `SUBROUTINE checkdat` - Checks that the input arrays are legal.

3. `SUBROUTINE checkposdef` - Checks that the information matrix is nonnegative definite.

4. `SUBROUTINE rsp` - calls `tred3` and `tqlrat` to find the eigenvalues of a real symmetric packed matrix. This subroutine comes from the eigensystem package EISPACK, [5]. The part of the original subroutine that is concerned with eigenvectors is omitted.

5. `SUBROUTINE tred3` - reduces a real symmetric matrix, stored as a one-dimensional array, to a symmetric tridiagonal matrix using orthogonal similarity transformations. This subroutine is a translation of the Algol procedure `tred3` in [6]. This subroutine comes from the eigensystem package EISPACK, [5].

6. `SUBROUTINE tqlrat` - finds the eigenvalues of a symmetric tridiagonal matrix by the rational QL method. This subroutine is a translation of the Algol procedure `tqlrat` in [7]. This subroutine comes from the eigensystem package EISPACK, [5]. Calls `epslon` and `pythag`.

7. `REAL*8 FUNCTION epslon` - estimates unit roundoff in quantities of a certain size. This function comes from the eigensystem package EISPACK, [5].

8. `REAL*8 FUNCTION pythag` - finds $\sqrt{a^2 + b^2}$ without overflow or destructive underflow. This function comes from the eigensystem package EISPACK, [5].

9. `SUBROUTINE calc_quota` - This is the shell of the iterative algorithm for computing the catch limit, see Section 3. Calls `putgauss` (Step 1). Calls `setSplit` (Step 2 and Step 3). Calls `halfInt` in order to compute approximations of I_{lower} and I_{upper} (Steps 4-9). Calls `zbrent` in order to find the zero of the function `fract` (Step 10).

10. `REAL*8 FUNCTION lhood` - Computes the scaled likelihood (the right-hand side of (4)) for a set of parameters.

11. `REAL*8 FUNCTION dens` - Integrates the scaled likelihood (the right-hand side of (4)) with respect to the bias parameter b. Multiplies the result by the Jacobi determinant of the transformation in (8). The result is a function of p_0 and μ. In the exceptional case when $p_0 < -5$ or $p_0 > 50$, the result is set to zero. Calls `pforw` in order to compute the population trajectory. Calls `evalgauss` in order to integrate `lhood`.

12. `SUBROUTINE pforw` - Computes the population size trajectory and $\frac{\partial P_T}{\partial P_0}$ for a set of parameters. In the ordinary case, this is done by using (1) and (9). In the exceptional case when $P_s < 10^{-30}$ for some s, P_t is set to 10^{-30} for $t = s, s+1, \ldots, T$. In the exceptional case when $P_0 > 2 \cdot 10^{10}$, the population size trajectory may not be accurately computed, and therefore the population size trajectory is computed in two ways. If the results are significantly different, this will be reported through the output value of the parameter `IFAIL` from the subroutine `CATCHLIMIT`.

13. `SUBROUTINE grule` - Computes the $[(n+1)/2]$ nonnegative abscissas x_i and corresponding weights w_i of the n-point Gauss-Legendre integration rule, normalized to the interval $[-1, 1]$, see Section 4.1.

14. `SUBROUTINE putgauss` - Sets up the coefficients for a n-point Gauss-Legendre integration rule for the b-integral. Calls `grule`.

15. `SUBROUTINE evalgauss` - Approximates a one-dimensional integral of a function using the Gauss-Legendre integration rule, see Section 4.1.

16. `SUBROUTINE prodgauss` - Sets up integration w.r.t. μ and p_0. This is Step 10b in the approximation procedure described in Section 3. Calls `grule` and then applies (38) to find the abscissas and the weights for the p_0-integration. The limits of the p_0-integrals are found by calling `getSplit` to get the value of $p_{0,split}(\mu)$ (defined by (15) and set by `setSplit`), and by calling `xbrent` to find $p_{0,z}(\mu)$ (defined by (27)). In this case `xbrent` finds the zero of `intLevel` for the appropriate choice of μ and D_T.

17. `SUBROUTINE halfgauss` - Sets up integration w.r.t. μ and p_0. This routine is used in Steps 5 and 8 in the approximation procedure described in Section 3. Calls `grule` to find the abscissas and the weights for the u- or v-integration, and then applies (38) to find the abscissas and the weights for the p_0-integration. The limits of the p_0-integrals are found by calling `getSplit` to get the value of $p_{0,split}(\mu)$ (defined by (15) and set by `setSplit`), and by calling `xbrent` to find $p_{0,lowmid}(\mu)$ (defined by (16)) or $p_{0,highmid}(\mu)$ (defined by (23)). In this case `xbrent` finds the zero of `logptoldt` for the appropriate choice of μ and D_T.

18. `SUBROUTINE evalpgauss` - Approximates a two-dimensional integral of a function using iterated integration and Gauss-Legendre rules (see Section 4.1) to evaluate the iterated integrals.

19. `REAL*8 FUNCTION logptoldt` - Computes $\ln(P_T) - \ln(P_0) - \ln(D_T)$. Calls `pforw` in order to compute P_T.

20. `REAL*8 FUNCTION intLevel` - Determines the internal catch limit (5) as a function of p_0. Calls `pforw` in order to compute P_T.

21. `REAL*8 FUNCTION xbrent` - Finds a zero of a function using Brent's method (see Section 4.2).

22. `REAL*8 FUNCTION zbrent` - Finds a zero of a function using Brent's method (see Section 4.2). Except for some additional parameters, this function is equal to the function `xbrent`. Two copies are needed in order to avoid recursion.

23. `REAL*8 FUNCTION getSplit` - Gets the value of $p_{0,split}(\mu)$ defined by (15) for a given μ.

24. `SUBROUTINE setSplit` - Find the abscissas and weights for the μ-integral. Determines and stores the split points ($p_{0,split}(\mu)$ defined by (15)) for each μ used as abscissa in the integration rule. Calls `grule` and then applies (38) to find the abscissas and the weights for the μ-integration. Calls `xbrent` in order to find the zero of `logptoldt` $(p_{0,split}(\mu))$.

25. `REAL*8 FUNCTION halfInt` - Calculates a semi-infinite integral, I_{lower} or I_{upper}, of the scaled likelihood (the right-hand side of (4)). Calls `halfgauss`. Calls `evalpgauss` in order to integrate `dens`.

26. `REAL*8 FUNCTION fract` - Calculates the cumulative probability of the internal catch limit at x. Subtracts the probability level α from the result. Calls `prodgauss`. Calls `evalpgauss` in order to integrate `dens`.

C Changes

Changes between versions of April 1999 and June 1999: In the version of April 1999, the variance covariance matrix of the logarithm of the abundance estimates was input. Moreover, this matrix was assumed to be diagonal and specified by a one- dimensional array containing the diagonal elements only. In the version of June 1999, however, the information matrix of the logarithm of the abundance estimates is input. This matrix does not need to be diagonal.

When `IN_INFOLEVEL` is positive, a warning message is printed if this matrix is not nonnegative definite. Due to numerical inaccuracy a singular matrix may be declared as not being nonnegative definite. In such cases, however, the magnitude of the lowest eigenvalue computed by the module is small. In order to guide the user, this eigenvalue is printed along with the warning message.

In the version of June 1999, D_1, D_2, and D_3 in (32-34) are printed if `IN_INFOLEVEL` is 6 or greater. In order to print likelihood and density values, `IN_INFOLEVEL` must be at least 7.

In the version of June 1999, `IFAIL = 6` on exit, means that the information matrix of the logarithm of the abundance estimates is not nonnegative definite.

Changes between versions of June 1999 and June 2000: In the version of June 2000, the sequence `AB_YEARS(I); I=1,...,ABDIM`; should be strictly increasing. This is checked in the subroutine `checkdat`.

In the version of June 2000, the upper bound of the interval in which the solution is seeked can be greater than in the version of June 1999. In the version of June 1999, the upper bound is $\mu_{\max} A_\tau$, where A_τ is the most recent abundance estimate. This bound could be too small if the variance of the most recent abundance estimate is large.

In the version of June 2000, the upper bound is $\mu_{\max} A_{\tau-1}$, where $A_{\tau-1}$ is the second most recent abundance estimate, provided that $A_\tau < A_{\tau-1}$, and the variance of the second most recent abundance estimate is smaller than the variance of the most recent abundance estimate. Otherwise, the upper bound is the same as in the version of June 1999.

Changes between versions of June 2000 and November 2000: The initial value of `last_quota` in `calc_quota` is changed from 0 to -10^{30} in order to avoid too early termination.

D Fortran code

```
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
C     xrmpSub.f
C
C     This module contains an implementation of
C     the catch limit algorithm.
C     Norwegian Computing Center, december 1992, Jon Helgeland
C     Modified, June 1999, Ragnar Bang Huseby
C     Modified, June 2000, Ragnar Bang Huseby
C     Modified, November 2000, Ragnar Bang Huseby

      SUBROUTINE CATCHLIMIT(NUM,ABDIM,CATCH,ABEST,INFOMATRX,R8WORKSPACE,
     *     AB_YEARS,IN_PPROB,IN_MU_MIN,IN_MU_MAX,IN_DT_MIN,IN_DT_MAX,
     *     IN_B_MIN,IN_B_MAX,IN_PLEVEL,
     *     OUT_QUOTA,accQuota,outDiff,npRule,
     *     POP,DEVPOP,IN_INFOLEVEL,IN_IOUT,IFAIL)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Calculate the catch limit for a single area according to the
C     algorithm described in [1].
C
C
C     Restrictions
C     ------------
C     We assume that the abundance estimates are positive, and
C     the information matrix of the logarithm of the estimates
C     is nonnegative definite. The subroutine is unable to compute
C     the catch limit in extreme cases.
C
C
C     Include files
C     -------------
C     xrmpSub_inc.f
C
C
C     Parameters
C     ----------
C      1. NUM - integer                                 Input
C              On entry: The length of the catch history.
```

```
C               Actual length of CATCH array
C
C      2. ABDIM - integer                               Input
C               On entry: Number of years with nonzero abundance
C               estimates for the area in question.
C
C      3. CATCH(NUM) - real array Input
C On entry: CATCH(Y) is the historic catch in year Y;
C               Y = 1,2,...,NUM.
C               Corresponds to Ct of (1) in [2].
C               If there has been any catch,
C               Y = 1 corresponds to the first year of catch.
C               Y = NUM corresponds to last premanagement year.
C               If there has been no catch, NUM should be equal to 1.
C               If NUM=1 and CATCH(1)=0, CATCH(1) is set to 1.
C               Constraint: CATCH(1) > 0 if NUM > 1.
C
C      4. ABEST(ABDIM) - real array Input
C On entry: ABEST(I) is the absolute abundance estimates
C               in year AB_YEARS(I); I = 1,2,...,ABDIM.
C               The vector of the logarithms of the entries in this
C               array corresponds to a of (2) in [2].
C               Constraint: ABEST(I) > 0.
C
C      5. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array Input
C On entry: INFOMATRX contains the lower triangle of
C               the matrix H in (2) in [2] stored row-wise.
C               Constraint: H is symmetric and nonnegative definite.
C
C      6. R8WORKSPACE(ABDIM*(ABDIM+7)/2) - real*8 array Workspace
C               Workspace needed to determine whether matrix stored
C               in INFOMATRX is nonnegative definite.
C
C      7. AB_YEARS(ABDIM) - integer array Input
C On entry: AB_YEARS(I) is the year of the absolute
C               abundance estimate ABEST(I).
C               If AB_YEARS(I) < 1, the corresponding abundance estimate
C               is treated as if the sighting was performed in year 1.
C               Constraints: AB_YEARS(I) < NUM+1.
C               AB_YEARS(1) < AB_YEARS(2) < ... < AB_YEARS(ABDIM)
C
C      8. IN_PPROB - real Input
C On entry: Probability level for distribution of Lt
C               Corresponds to alpha of (6) in [2].
```

```
C               Typical value: 0.4102
C
C        9. IN_MU_MIN - real Input
C On entry: Minimum value of productivity parameter
C               See (3) in [2].
C               Typical value: 0.0
C               Constraint: IN_MU_MIN is nonnegative.
C
C       10. IN_MU_MAX - real Input
C           On entry: Maximum value of productivity parameter
C               See (3) in [2].
C               Typical value: 0.05
C               Constraints: IN_MU_MAX is not less than 10**(-20).
C                            IN_MU_MAX is not less than IN_MU_MIN.
C
C       11. IN_DT_MIN - real Input
C On entry: Minimum value for stock depletion
C               See (3) in [2].
C               Typical value: 0.0
C               Constraint: IN_DT_MIN is nonnegative.
C
C       12. IN_DT_MAX - real Input
C On entry: Maximum value for stock depletion
C               See (3) in [2].
C               Typical value: 1.0
C               Constraint: IN_DT_MAX is not less than IN_DT_MIN.
C
C       13. IN_B_MIN - real Input
C On entry: Minimum bias
C               See (3) in [2].
C               Typical value: 0.0
C               Constraint: IN_B_MIN is nonnegative.
C
C       14. IN_B_MAX - real Input
C On entry: Maximum bias
C               See (3) in [2].
C               Typical value: 1.6667
C               Constraints: IN_B_MAX is not less than 10**(-20).
C                            IN_B_MAX is not less than IN_B_MIN.
C
C       15. IN_PLEVEL - real Input
C On entry: Internal protection level
C               Corresponds to IPL of (5) in [2].
C               Typical value: 0.54
```

```
C              Constraint: IN_PLEVEL is in [IN_DT_MIN,IN_DT_MAX].
C
C     16. OUT_QUOTA - real Output
C On exit: Calculated catch limit
C
C     17. accQuota - real*8 Input
C On entry: Tolerance specifying the required accuracy.
C              The iterative algorithm terminates if the difference
C              between two successive approximations of the catch limit
C              (determined by n/2-point and n-point Gauss-Legendre
C              integration rules, respectively) is less or equal to
C              accQuota.
C              The approximate solution of (29) in [2] is determined
C              such that its accuracy is 0.25 * accQuota.
C              Typical value: 0.2
C
C     18. outDiff - real*8 Output
C On exit: Achieved accuracy, that is the difference
C              between the last two approximations of the catch limit.
C
C     19. npRule - integer Output
C On exit: The number of points used in the numerical
C              integration in the last iteration.
C
C     20. POP(0:NUM+1) - real*8 array                Workspace
C              Various population size trajectories.
C              Corresponds to Pt of (1) in [2].
C
C     21. DEVPOP(ABDIM) - real*8 array         Workspace
C              Difference between abundance estimate and population
C              size at years with abundance estimates for various
C              trajectories.
C
C     22. IN_INFOLEVEL - integer Input
C On entry: Parameter controlling the level of
C              intermediate printout produced by this module.
C              The larger value, the more printout.
C              Typical values:
C              0 - no printout
C              1 - possible warnings
C              2 - as 1 + print each catch limit approximation
C              3 - as 2 + print value of integrals
C              4 - as 3 + print some integration limits
C              5 - as 4 + print input arrays
```

```
C               6 - as 5 + print D1, D2, and D3 in (32-34) in [2].
C               7 - as 6 + print likelihood and density values
C
C     23. IN_IOUT - integer Input
C On entry: Unit determining file for intermediate
C               printout.
C
C     24. IFAIL - integer Input/Output
C On entry: If the user sets IFAIL to 0 before calling
C the routine, execution of the program will terminate
C if the routine detects an error. Before the program is
C stopped, an error message is output.
C               If the user sets IFAIL to -1 or 1 before calling the
C routine, the control is returned to the calling
C program if the routine detecs an error.
C               If IFAIL = -1, an error message is output before the
C               control is returned.
C On exit: If IFAIL = 0, no error is detected.
C               If IFAIL = 2, NUM < 1.
C               If IFAIL = 3, ABDIM < 1.
C               If IFAIL = 4, NUM > 1 and CATCH(1) is not positive.
C               If IFAIL = 5, ABEST(I) is not positive for some I.
C               If IFAIL = 6, the information matrix of the logarithm of
C               the abundance estimates is not nonnegative definite (At
C               least one of the eigenvalues is negative).
C               Due to numerical inaccuracy a singular matrix may be
C               declared as not being nonnegative definite. In such cases,
C               however, the magnitude of the lowest eigenvalue computed
C               by the module is small. This eigenvalue is printed
C               if IN_INFOLEVEL is positive.
C               If IFAIL = 7, AB_YEARS(I) > NUM for some I, or the sequence
C               AB_YEARS(I); I=1,...,ABDIM; is not strictly increasing.
C               If IFAIL = 8, IN_PPROB < 0 or IN_PPROB > 1.
C               If IFAIL = 9, MU_MIN > MU_MAX, MU_MIN < 0,
C               or MU_MAX < 10**(-20).
C               If IFAIL = 10, DT_MIN > DT_MAX or DT_MIN < 0.
C               If IFAIL = 11, B_MIN > B_MAX, B_MIN < 0,
C               or B_MAX < 10**(-20).
C               If IFAIL = 12, IN_PLEVEL < DT_MIN or IN_PLEVEL > DT_MAX.
C               If IFAIL = 13, accQuota is not positive.
C               If IFAIL = 14, 'nmax' in include file is less than the
C number of rule points.
C               If IFAIL = 15, possible inaccuracies in computed
C               population size history.
```

```
C               If IFAIL = 16, PT becomes larger than 0.5*10**30.
C               If IFAIL = 17, the Jacobi determinant ((7) in [2])
C               becomes negative.
C               If IFAIL = 18, for some mu it was not possible to find
C               p0 such that DT = IN_PLEVEL.
C               If IFAIL = 19, for some mu it was not possible to find
C               p0 such that either (14) or (21) in [2] is satisfied.
C               If IFAIL = 20, for some mu it was not possible to find
C               the integration interval of the p0-integral.
C               If IFAIL = 21, the value of z in (24) in [2] becomes
C               negative.
C               If IFAIL = 22, the catch limit could not be computed
C               because the computed approximation of (10) in [2] is
C               zero.
C               If IFAIL = 23, it was not possible to solve the equation
C               for the catch limit.
C               If IFAIL = 24, the required accuracy was not reached.
C               If IFAIL = -2, the input value of IFAIL is illegal. It
C               is assumed that IFAIL value should be 0.
C
C
C     Authors
C     -------
C     Rolf Volden and Jon Helgeland,
C     Modified december,1992 by Jon Helgeland
C     Norwegian Computing Center (NR) 1992
C     Modified by: Ragnar Bang Huseby, April 1999
C     Additional modifications by: Ragnar Bang Huseby, June 1999
C     Additional modifications by: Ragnar Bang Huseby, June 2000
C
C
C     Changes in CATCHLIMIT between versions of April 1999 and June 1999:
C     -------------------------------------------------------------------
C     In the version of April 1999, the variance covariance matrix of
C     the logarithm of the abundance estimates was input. Moreover,
C     this matrix was assumed to be diagonal and specified by a one-
C     dimensional array containing the diagonal elements only.
C     In the version of June 1999, however, the information matrix of
C     the logarithm of the abundance estimates is input. This matrix
C     does not need to be diagonal.
C
C     When IN_INFOLEVEL is positive, a warning message is printed if this
C     matrix is not nonnegative definite. Due to numerical inaccuracy
C     a singular matrix may be declared as not being nonnegative
```

```
C     definite. In such cases, however, the magnitude of the lowest
C     eigenvalue computed by the module is small. In order to guide the
C     user, this eigenvalue is printed along with the warning message.
C
C     In the version of June 1999, D1, D2, and D3 in (32-34) in [2]
C     are printed if IN_INFOLEVEL is 6 or greater. In order to print
C     likelihood and density values, IN_INFOLEVEL must be at least 7.
C
C     In the version of June 1999, IFAIL = 6 on exit, means that
C     the information matrix of the logarithm of the abundance estimates
C     is not nonnegative definite.
C
C
C     Changes in CATCHLIMIT between versions of June 1999 and June 2000:
C     -------------------------------------------------------------------
C     In the version of June 2000, the sequence AB_YEARS(I); I=1,...,ABDIM;
C     should be strictly increasing. This is checked in the subroutine
C     'checkdat'.
C
C     In the version of June 2000, the upper bound of the interval in which
C     the solution is seeked can be greater than in the version of June 1999.
C     In the version of June 1999, the upper bound is MU_MAX*ABEST(ABDIM).
C     This bound could be too small if the variance of the most recent
C     abundance estimate is large.
C
C     In the version of June 2000, the upper bound is MU_MAX*ABEST(ABDIM-1)
C     provided that ABEST(ABDIM) < ABEST(ABDIM-1), and the variance of the
C     second most recent abundance estimate is smaller than the variance of
C     the most recent abundance estimate. Otherwise, the upper bound is the
C     same as in the version of June 1999.
C
C     Changes between versions of June 2000 and November 2000:
C     ---------------------------------------------------------
C     The initial value of last_quota in calc_quota is changed
C     from 0 to -10**30 in order to avoid too early termination.
C
C
C     References
C     ----------
C     [1] Rep. Int. Whal. Commn. 44, 1994, Annex H.
C
C     [2] Huseby, R.B. and Aldrin, M.:
C     "Documentation of a Fortran 77 subroutine implementing
C     the catch limit algorithm", Note no SAMBA/25/2000,
```

```
C     Norwegian Computing Center, 2000.
C
C     [3] Davis, P.J. and Rabinowitz P.:
C     "Methods of numerical integration",
C     Academic Press, 1975.
C
C     [4] Martin, Reinsch, and Wilkinson:
C     "Num. Math. 11", pp 181-195, 1968.
C
C     [5] Reinsch:
C     "Comm. ACM 16", p 689, 1973.
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM)
      REAL INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 R8WORKSPACE(ABDIM*(ABDIM+7)/2)
      REAL OUT_QUOTA
      REAL*8 accQuota,outDiff
      INTEGER npRule
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)
      REAL IN_PPROB,IN_MU_MIN,IN_MU_MAX,IN_DT_MIN,IN_DT_MAX,
     *     IN_B_MIN,IN_B_MAX,IN_PLEVEL
      INTEGER IN_INFOLEVEL,IN_IOUT,IFAIL

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      REAL PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,PLEVEL
      COMMON /MANPAR/ PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,
     *     PLEVEL

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      REAL EPS
      INTEGER failOption, defStatus
```

```
C-----------------------------------------------------------------------

      DATA EPS /1.0E-20/
      defStatus = 0

C     Transfer from input parameters to common variables
      PPROB=IN_PPROB
      MU_MIN=IN_MU_MIN
      MU_MAX=IN_MU_MAX
      DT_MIN=IN_DT_MIN
      DT_MAX=IN_DT_MAX
      B_MIN=IN_B_MIN
      B_MAX=IN_B_MAX
      PLEVEL=IN_PLEVEL
      infoLevel=IN_INFOLEVEL
      IOUT = IN_IOUT

      if (infoLevel.ge.1)then
         write(IOUT,*) ' '
         write(IOUT,*) '************************************************
     ************************'
         write(IOUT,*) '  Starting routine CATCHLIMIT'
         write(IOUT,*) '------------------------------------------------
     *-----------------------'
      endif

      failOption = IFAIL
      failStatus = OK
      if (abs(failOption).gt.1) then
         failStatus = IFAILERR
         failOption = 0
         goto 10
      endif

C     Control input data
      call checkdat(NUM,ABDIM,CATCH,ABEST,INFOMATRX,R8WORKSPACE,
     *     AB_YEARS, defStatus)
      if (failStatus .ne. OK) GOTO 10
      if (accQuota.le.0.0D0) failStatus = ACCQUOTAERR
      if (PLEVEL.lt.DT_MIN.or.PLEVEL.gt.DT_MAX) failStatus = PLEVELERR
      if (B_MIN.gt.B_MAX) failStatus = BERR
      if (B_MIN.lt.0.0) failStatus = BERR
      if (B_MAX.lt.EPS) failStatus = BERR
      if (B_MIN  .LE. 0.0) B_MIN  = EPS
```

```
      if (DT_MIN.gt.DT_MAX) failStatus = DTERR
      if (DT_MIN.lt.0.0) failStatus = DTERR
      if (MU_MIN.gt.MU_MAX) failStatus = MUERR
      if (MU_MIN.lt.0.0) failStatus = MUERR
      if (MU_MAX.lt.EPS) failStatus = MUERR
      if (MU_MIN .LE. 0.0) MU_MIN = EPS
      if (PPROB.lt.0.0D0.or.PPROB.gt.1.0D0) failStatus = PPROBERR
      if (failStatus .ne. OK) GOTO 10

C     Calculate catch limit
      call calc_quota(OUT_QUOTA,accQuota,outDiff,npRule,
     *     NUM,ABDIM,CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)

      if (defStatus .eq. 1) failStatus = VERR

 10   IFAIL = failStatus

      if (infoLevel.ge.1)then
         write(IOUT,*) '-----------------------------------------------
     *-----------------------'
         write(IOUT,*) '  Finishing routine CATCHLIMIT'
         write(IOUT,*) '***********************************************
     ***********************'
         write(IOUT,*) ' '
      endif

      if (failStatus.eq.OK .or. failOption.eq.1) RETURN
      WRITE(*,*) '** ABNORMAL EXIT from routine CATCHLIMIT: IFAIL =',
     *     IFAIL
      if (failOption.eq.0) then
         WRITE(*,*) '** Hard failure - execution terminated'
         STOP
      endif
      WRITE(*,*) '** Soft failure - control returned'
      RETURN
      END
C-----------------------------------------------------------------------



      SUBROUTINE checkdat(NUM,ABDIM,CATCH,ABEST,INFOMATRX,R8WORKSPACE,
     *     AB_YEARS,defStatus)
C-----------------------------------------------------------------------
```

```
C
C     Purpose
C     -------
C     Check that the input is legal.
C
C     Parameters
C     -----------
C      1. NUM - integer       Input
C See CATCHLIMIT
C
C      2. ABDIM  - integer       Input
C See CATCHLIMIT
C
C      3. CATCH(NUM) - real array Input
C See CATCHLIMIT
C
C      4. ABEST(ABDIM) - real array Input
C See CATCHLIMIT
C
C      5. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array Input
C See CATCHLIMIT
C
C      6. R8WORKSPACE(ABDIM*(ABDIM+7)/2) - real*8 array Workspace
C See CATCHLIMIT
C
C      7. AB_YEARS(ABDIM) - integer array Output
C See CATCHLIMIT
C
C      8. defStatus - integer                  Output
C On exit: If defStatus = 0, the information matrix of
C              the logarithm of the abundance estimates is nonnegative
C              definite. If defStatus = 1, this matrix is not
C              nonnegative definite.
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER NUM,ABDIM
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 R8WORKSPACE(ABDIM*(ABDIM+7)/2)
      INTEGER AB_YEARS(ABDIM)
      INTEGER defStatus
```

```
C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER I,J
      INTEGER ISTART_A,ISTART_W,ISTART_FV1,ISTART_FV2,
     *     IEND_A,IEND_W,IEND_FV1,IEND_FV2
C-----------------------------------------------------------------------

C     Check that the length the array containing the catch history
C     is at least one.
      if (NUM.lt.1 .and. failStatus.eq.OK) failStatus=NUMERR

C     Check that the length the array containing the abundance estimates
C     is at least one.
      if (ABDIM.lt.1 .and. failStatus.eq.OK) failStatus=ABDIMERR

C     If no catch, CATCH(1) = 1.0
      if (NUM.eq.1 .and. CATCH(1).LE.0.0) CATCH(1) = 1.0

C     Check that there has been a catch in the first year
      if (CATCH(1).LE.0.0 .and. failStatus.eq.OK) failStatus=CATCHERR

C     Check that the abundance estimates are positive
      if (failStatus.eq.OK) then
         DO 10 I=1,ABDIM
            if (ABEST(I) .LE. 0.0) failStatus = AERR
 10      CONTINUE
      endif

      if (failStatus.ne.OK) RETURN

C     In later subroutines:
C     A(NV) is stored in R8WORKSPACE(ISTART_A:IEND_A)
C     W(N) is stored in R8WORKSPACE(ISTART_W:IEND_W)
C     FV1(N) is stored in R8WORKSPACE(ISTART_FV1:IEND_FV1)
C     FV2(N) is stored in R8WORKSPACE(ISTART_FV2:IEND_FV2)
      ISTART_A = 1
```

```
      IEND_A = ABDIM*(ABDIM+1)/2
      ISTART_W = IEND_A+1
      IEND_W = IEND_A+ABDIM
      ISTART_FV1 = IEND_W+1
      IEND_FV1 = IEND_W+ABDIM
      ISTART_FV2 = IEND_FV1+1
      IEND_FV2 = IEND_FV1+ABDIM

C     Check that the information matrix is nonnegative definite
      call checkposdef(ABDIM,INFOMATRX,
     *     R8WORKSPACE(ISTART_A),R8WORKSPACE(ISTART_W),
     *     R8WORKSPACE(ISTART_FV1),R8WORKSPACE(ISTART_FV2), defStatus)

C     Test if some abundance estimate is prior to year of first catch
C     Make sure that AB_YEARS(I); I=1,...,ABDIM; is a strictly increasing
C     sequence
      DO 30 I=1,ABDIM
         J=AB_YEARS(I)
         if(J.lt.0 .and. infoLevel.ge.1)then
            write(IOUT,*)
     *  '**** Warning: Abundance estimate prior to year of first catch',
     *  'treated as first year'
         endif
         if(J.gt.NUM) then
            failStatus = ABYEARERR
            RETURN
         endif
         if(I.gt.1) then
            if(j.le.AB_YEARS(I-1)) then
               failStatus = ABYEARERR
               RETURN
            endif
         endif
 30      CONTINUE

      if(infoLevel.ge.5)then
         WRITE(IOUT,*) ' ABDIM,AB_YEARS: ',ABDIM,AB_YEARS(1)
      endif

      if (infoLevel.ge.5)then
         write(IOUT,*) 'Catch history:'
         do I=1,NUM
            write(IOUT,*) ' YEAR, CATCH: ', I, CATCH(I)
         enddo
```

```
         write(IOUT,*) 'Abundance estimates:'
         do I=1,ABDIM
            write(IOUT,*) ' YEAR, ABEST: ',
     *           AB_YEARS(I), ABEST(I)
         enddo
         write(IOUT,*) 'Information matrix:'
         do I=1,ABDIM
            write(IOUT,*) (INFOMATRX((I*(I-1))/2+J),J=1,I)
         enddo
         write(IOUT,*) ' '
      endif
      RETURN

      END
C-----------------------------------------------------------------------



      SUBROUTINE checkposdef(n,SYMMATRX,A,W,FV1,FV2,defStatus)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     This subroutine checks whether a symmetric matrix A is
C     nonnegative definite. This is done by computing the eigenvalues
C     of A, and checking that they are all nonnegative.
C
C     Parameters
C     ----------
C      1. n - integer Input
C On entry: The number of rows (= the number of columns)
C                         of A.
C
C      2. SYMMATRX - real*8 array Input
C On entry: SYMMATRX contains the lower triangle of A
C               stored row-wise.
C
C      3. A - real array                                Input
C               On entry: THE LOWER TRIANGLE OF THE REAL SYMMETRIC
C               PACKED MATRIX STORED ROW-WISE.
C
C      4. W - real array                                Output
C               On exit: THE EIGENVALUES IN ASCENDING ORDER.
```

```
C
C      5. FV1 - real array                                 Workspace
C
C      6. FV2 - real array                                 Workspace
C
C      7. defStatus - integer                  Output
C See checkdat
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL SYMMATRX(n*(n+1)/2)
      REAL*8 A(n*(n+1)/2),W(n),FV1(n),FV2(n)
      INTEGER n, defStatus

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER k,m,IERR
C-----------------------------------------------------------------------

C     Copy from SYMMATRX to A
      m=n*(n+1)/2
      do k=1,m
        A(k)=SYMMATRX(k)
      enddo

C     Compute the eigenvalues of A
      CALL rsp(n,A,W,FV1,FV2,IERR)

      if ((IERR.ne.0).and.(infoLevel.ge.1)) then
         write(IOUT,*)
     *        ' **** Warning: Not able to determine all
     *        the eigenvalues of the information matrix.'
      endif
      if ((W(1).lt.0.0D0).and.(infoLevel.ge.1)) then
```

```
         defStatus = 1
         write(IOUT,*)
     *        ' **** Warning: The information matrix may not be nonnegat
     *ive definite.',
     *        '              The lowest eigenvalue is approximately',
     *        W(1)
      endif
      return
      end
C-----------------------------------------------------------------------



      SUBROUTINE rsp(N,A,W,FV1,FV2,IERR)
C------------------------------------------------------------------------
C
C     Purpose
C     -------
C     THIS SUBROUTINE CALLS THE RECOMMENDED SEQUENCE OF
C     SUBROUTINES FROM THE EIGENSYSTEM SUBROUTINE PACKAGE (EISPACK)
C     TO FIND THE EIGENVALUES OF A REAL SYMMETRIC PACKED MATRIX.
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO BURTON S. GARBOW,
C     MATHEMATICS AND COMPUTER SCIENCE DIV, ARGONNE NATIONAL LABORATORY
C
C     THIS VERSION DATED AUGUST 1983.
C
C
C     Parameters
C     ----------
C      1. N - integer                                    Input
C               On entry: THE ORDER OF THE MATRIX  A.
C
C      2. A - real array                                 Input
C               On entry: THE LOWER TRIANGLE OF THE REAL SYMMETRIC
C               PACKED MATRIX STORED ROW-WISE.
C
C      3. W - real array                                 Output
C               On exit: THE EIGENVALUES IN ASCENDING ORDER.
C
C      4. FV1 - real array                               Workspace
C
C      5. FV2 - real array                               Workspace
```

```
C
C      6. IERR - integer                                     Output
C               On exit: INTEGER OUTPUT VARIABLE SET
C               EQUAL TO AN ERROR COMPLETION CODE DESCRIBED IN
C               THE DOCUMENTATION FOR tqlrat.
C               THE NORMAL COMPLETION CODE IS ZERO.
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER N,IERR
      REAL*8 A(N*(N+1)/2),W(N),FV1(N),FV2(N)
C
C     LOCAL DEFINITIONS
C
      INTEGER NV
C-----------------------------------------------------------------------

      NV = (N * (N + 1)) / 2

C     Reduce symmetric matrix to a tridiagonal matrix
      CALL  tred3(N,NV,A,W,FV1,FV2)

C     Find eigenvalues
      CALL  tqlrat(N,W,FV2,IERR)
      RETURN
      END
C-----------------------------------------------------------------------




      SUBROUTINE tred3(N,NV,A,D,E,E2)
C------------------------------------------------------------------------
C
C     Purpose
C     -------
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE tred3, [4].
C
C     THIS SUBROUTINE REDUCES A REAL SYMMETRIC MATRIX, STORED AS
C     A ONE-DIMENSIONAL ARRAY, TO A SYMMETRIC TRIDIAGONAL MATRIX
C     USING ORTHOGONAL SIMILARITY TRANSFORMATIONS.
```

```
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO BURTON S. GARBOW,
C     MATHEMATICS AND COMPUTER SCIENCE DIV, ARGONNE NATIONAL LABORATORY
C
C     THIS VERSION DATED AUGUST 1983.
C
C     Parameters
C     ----------
C
C      1. N - integer                                    Input
C              On entry: THE ORDER OF THE MATRIX.
C
C      2. NV - integer                                   Input
C              On entry: THE DIMENSION OF THE ARRAY PARAMETER A
C
C      3. A - real array                                 Input/Output
C              On entry: THE LOWER TRIANGLE OF THE REAL SYMMETRIC
C              INPUT MATRIX, STORED ROW-WISE AS A ONE-DIMENSIONAL
C              ARRAY, IN ITS FIRST N*(N+1)/2 POSITIONS.
C              On exit: INFORMATION ABOUT THE ORTHOGONAL
C              TRANSFORMATIONS USED IN THE REDUCTION.
C
C      4. D - real array
C              On exit: THE DIAGONAL ELEMENTS OF THE TRIDIAGONAL MATRIX.
C
C      5. E - real array
C              On exit: THE SUBDIAGONAL ELEMENTS OF THE TRIDIAGONAL
C              MATRIX IN ITS LAST N-1 POSITIONS.  E(1) IS SET TO ZERO.
C
C      6. E2 - real array
C              On exit: THE SQUARES OF THE CORRESPONDING ELEMENTS OF E.
C              E2 MAY COINCIDE WITH E IF THE SQUARES ARE NOT NEEDED.
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER N,NV
      REAL*8 A(NV),D(N),E(N),E2(N)

C     LOCAL DEFINITIONS
C
      INTEGER I,J,K,L,II,IZ,JK,JM1
```

```
      REAL*8 F,G,H,HH,SCALE
C-------------------------------------------------------------------------
C
C     .......... FOR I=N STEP -1 UNTIL 1 DO -- ..........
      DO 300 II = 1, N
         I = N + 1 - II
         L = I - 1
         IZ = (I * L) / 2
         H  = 0.0D0
         SCALE = 0.0D0
         IF (L .LT. 1) GO TO 130
C     .......... SCALE ROW (ALGOL TOL THEN NOT NEEDED) ..........
         DO 120 K = 1, L
            IZ = IZ + 1
            D(K) = A(IZ)
            SCALE = SCALE + DABS(D(K))
 120     CONTINUE
C
         IF (SCALE .NE. 0.0D0) GO TO 140
 130     E(I) = 0.0D0
         E2(I) = 0.0D0
         GO TO 290
C
 140     DO 150 K = 1, L
            D(K) = D(K) / SCALE
            H = H + D(K) * D(K)
 150     CONTINUE
C
         E2(I) = SCALE * SCALE * H
         F = D(L)
         G = -DSIGN(DSQRT(H),F)
         E(I) = SCALE * G
         H = H - F * G
         D(L) = F - G
         A(IZ) = SCALE * D(L)
         IF (L .EQ. 1) GO TO 290
         JK = 1
C
         DO 240 J = 1, L
            F = D(J)
            G = 0.0D0
            JM1 = J - 1
            IF (JM1 .LT. 1) GO TO 220
C
```

```
            DO 200 K = 1, JM1
               G = G + A(JK) * D(K)
               E(K) = E(K) + A(JK) * F
               JK = JK + 1
 200        CONTINUE
C
 220        E(J) = G + A(JK) * F
            JK = JK + 1
 240     CONTINUE
C     .......... FORM P ..........
         F = 0.0D0
C
         DO 245 J = 1, L
            E(J) = E(J) / H
            F = F + E(J) * D(J)
 245     CONTINUE
C
         HH = F / (H + H)
C     .......... FORM Q ..........
         DO 250 J = 1, L
 250        E(J) = E(J) - HH * D(J)
C
            JK = 1
C     .......... FORM REDUCED A ..........
            DO 280 J = 1, L
               F = D(J)
               G = E(J)
C
               DO 260 K = 1, J
                  A(JK) = A(JK) - F * E(K) - G * D(K)
                  JK = JK + 1
 260           CONTINUE
C
 280        CONTINUE
C
 290        D(I) = A(IZ+1)
            A(IZ+1) = SCALE * DSQRT(H)
 300     CONTINUE
C
         RETURN
         END
C-----------------------------------------------------------------------
```

```
      SUBROUTINE tqlrat(N,D,E2,IERR)
C-------------------------------------------------------------------------
C
C     Purpose
C     -------
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE tqlrat, [5]
C     (ALGORITHM 464).
C
C     THIS SUBROUTINE FINDS THE EIGENVALUES OF A SYMMETRIC
C     TRIDIAGONAL MATRIX BY THE RATIONAL QL METHOD.
C
C     CALLS pythag FOR  DSQRT(A*A + B*B) .
C
C     QUESTIONS AND COMMENTS SHOULD BE DIRECTED TO BURTON S. GARBOW,
C     MATHEMATICS AND COMPUTER SCIENCE DIV, ARGONNE NATIONAL LABORATORY
C
C     THIS VERSION DATED AUGUST 1987.
C     MODIFIED BY C. MOLER TO FIX UNDERFLOW/OVERFLOW DIFFICULTIES,
C     ESPECIALLY ON THE VAX AND OTHER MACHINES WHERE EPSLON(1.0D0)**2
C     NEARLY UNDERFLOWS.  SEE THE LOOP INVOLVING STATEMENT 102 AND
C     THE TWO STATEMENTS JUST BEFORE STATEMENT 200.
C
C
C     Parameters
C     ----------
C
C      1. N - integer                                  Input
C              On entry: THE ORDER OF THE MATRIX.
C
C      2. D - real array                               Input/Output
C              On entry: THE DIAGONAL ELEMENTS OF THE INPUT MATRIX.
C              On exit: THE EIGENVALUES IN ASCENDING ORDER.  IF AN
C              ERROR EXIT IS MADE, THE EIGENVALUES ARE CORRECT AND
C              ORDERED FOR INDICES 1,2,...IERR-1, BUT MAY NOT BE
C              THE SMALLEST EIGENVALUES.
C      3. E2 - real array                              Input/Output
C              On entry: THE SQUARES OF THE SUBDIAGONAL ELEMENTS OF
C              THE INPUT MATRIX IN ITS LAST N-1 POSITIONS.
C              E2(1) IS ARBITRARY.
C              On exit: DESTROYED
C      4. IERR - integer                               Output
C              On exit: If IERR = 0, NORMAL RETURN,
```

```
C              IF = J, THE J-TH EIGENVALUE HAS NOT BEEN
C              DETERMINED AFTER 30 ITERATIONS.
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER N,IERR
      REAL*8 D(N),E2(N)

C     LOCAL DEFINITIONS
C
      INTEGER I,J,L,M,II,L1,MML
      REAL*8 B,C,F,G,H,P,R,S,T,epslon,pythag
C-----------------------------------------------------------------------

      IERR = 0
      IF (N .EQ. 1) GO TO 1001
C
      DO 100 I = 2, N
  100 E2(I-1) = E2(I)
C
      F = 0.0D0
      T = 0.0D0
      E2(N) = 0.0D0
C
      DO 290 L = 1, N
         J = 0
         H = DABS(D(L)) + DSQRT(E2(L))
         IF (T .GT. H) GO TO 105
         T = H
         B = EPSLON(T)
         C = B * B
         IF (C .NE. 0.0D0) GO TO 105
C        SPLITING TOLERANCE UNDERFLOWED.  LOOK FOR LARGER VALUE.
         DO 102 I = L, N
            H = DABS(D(I)) + DSQRT(E2(I))
            IF (H .GT. T) T = H
  102    CONTINUE
         B = EPSLON(T)
         C = B * B
C     .......... LOOK FOR SMALL SQUARED SUB-DIAGONAL ELEMENT ..........
  105    DO 110 M = L, N
            IF (E2(M) .LE. C) GO TO 120
```

```
C     .......... E2(N) IS ALWAYS ZERO, SO THERE IS NO EXIT
C                THROUGH THE BOTTOM OF THE LOOP ..........
  110    CONTINUE
C
  120    IF (M .EQ. L) GO TO 210
  130    IF (J .EQ. 30) GO TO 1000
         J = J + 1
C     .......... FORM SHIFT ..........
         L1 = L + 1
         S = DSQRT(E2(L))
         G = D(L)
         P = (D(L1) - G) / (2.0D0 * S)
         R = PYTHAG(P,1.0D0)
         D(L) = S / (P + DSIGN(R,P))
         H = G - D(L)
C
         DO 140 I = L1, N
  140    D(I) = D(I) - H
C
         F = F + H
C     .......... RATIONAL QL TRANSFORMATION ..........
         G = D(M)
         IF (G .EQ. 0.0D0) G = B
         H = G
         S = 0.0D0
         MML = M - L
C     .......... FOR I=M-1 STEP -1 UNTIL L DO -- ..........
         DO 200 II = 1, MML
            I = M - II
            P = G * H
            R = P + E2(I)
            E2(I+1) = S * R
            S = E2(I) / R
            D(I+1) = H + S * (H + D(I))
            G = D(I) - E2(I) / G
C           AVOID DIVISION BY ZERO ON NEXT PASS
            IF (G .EQ. 0.0D0) G = EPSLON(D(I))
            H = G * (P / R)
  200    CONTINUE
C
         E2(L) = S * G
         D(L) = H
C     .......... GUARD AGAINST UNDERFLOW IN CONVERGENCE TEST ..........
         IF (H .EQ. 0.0D0) GO TO 210
```

```
         IF (DABS(E2(L)) .LE. DABS(C/H)) GO TO 210
         E2(L) = H * E2(L)
         IF (E2(L) .NE. 0.0D0) GO TO 130
  210    P = D(L) + F
C     .......... ORDER EIGENVALUES ..........
         IF (L .EQ. 1) GO TO 250
C     .......... FOR I=L STEP -1 UNTIL 2 DO -- ..........
         DO 230 II = 2, L
            I = L + 2 - II
            IF (P .GE. D(I-1)) GO TO 270
            D(I) = D(I-1)
  230    CONTINUE
C
  250    I = 1
  270    D(I) = P
  290 CONTINUE
C
      GO TO 1001
C     .......... SET ERROR -- NO CONVERGENCE TO AN
C                EIGENVALUE AFTER 30 ITERATIONS ..........
 1000 IERR = L
 1001 RETURN
      END
C-----------------------------------------------------------------------




      REAL*8 FUNCTION epslon (X)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     ESTIMATE UNIT ROUNDOFF IN QUANTITIES OF SIZE X.
C     THIS PROGRAM SHOULD FUNCTION PROPERLY ON ALL SYSTEMS
C     SATISFYING THE FOLLOWING TWO ASSUMPTIONS,
C        1.  THE BASE USED IN REPRESENTING FLOATING POINT
C            NUMBERS IS NOT A POWER OF THREE.
C        2.  THE QUANTITY  A  IN STATEMENT 10 IS REPRESENTED TO
C            THE ACCURACY USED IN FLOATING POINT VARIABLES
C            THAT ARE STORED IN MEMORY.
C     THE STATEMENT NUMBER 10 AND THE GO TO 10 ARE INTENDED TO
C     FORCE OPTIMIZING COMPILERS TO GENERATE CODE SATISFYING
C     ASSUMPTION 2.
```

```
C     UNDER THESE ASSUMPTIONS, IT SHOULD BE TRUE THAT,
C            A  IS NOT EXACTLY EQUAL TO FOUR-THIRDS,
C            B  HAS A ZERO FOR ITS LAST BIT OR DIGIT,
C            C  IS NOT EXACTLY EQUAL TO ONE,
C            EPS  MEASURES THE SEPARATION OF 1.0 FROM
C                 THE NEXT LARGER FLOATING POINT NUMBER.
C     THE DEVELOPERS OF EISPACK WOULD APPRECIATE BEING INFORMED
C     ABOUT ANY SYSTEMS WHERE THESE ASSUMPTIONS DO NOT HOLD.
C
C     THIS VERSION DATED 4/6/83.
C
C
C     Parameter
C     ---------
C      1. X - REAL*8                          Input
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 X
C
C     LOCAL DEFINITIONS
C
      REAL*8 A,B,C,EPS
C
C-----------------------------------------------------------------------
      A = 4.0D0/3.0D0
 10   B = A - 1.0D0
      C = B + B + B
      EPS = DABS(C-1.0D0)
      IF (EPS .EQ. 0.0D0) GO TO 10
      epslon = EPS*DABS(X)
      RETURN
      END
C-----------------------------------------------------------------------




      REAL*8 FUNCTION pythag(A,B)
C-----------------------------------------------------------------------
C
```

```
C     Purpose
C     -------
C     FINDS DSQRT(A**2+B**2) WITHOUT OVERFLOW OR DESTRUCTIVE UNDERFLOW
C
C
C     Parameters
C     ----------
C      1. A - REAL*8                          Input
C
C      2. B - REAL*8                          Input
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 A,B
C
C     LOCAL DEFINITIONS
C
      REAL*8 P,R,S,T,U
C
C-----------------------------------------------------------------------
      P = DMAX1(DABS(A),DABS(B))
      IF (P .EQ. 0.0D0) GO TO 20
      R = (DMIN1(DABS(A),DABS(B))/P)**2
 10   CONTINUE
      T = 4.0D0 + R
      IF (T .EQ. 4.0D0) GO TO 20
      S = R/T
      U = 1.0D0 + 2.0D0*S
      P = U*P
      R = (S/U)**2 * R
      GO TO 10
 20   pythag = P
      RETURN
      END
C-----------------------------------------------------------------------


      SUBROUTINE calc_quota(OUT_QUOTA,accQuota,outDiff,npRule,
     *     NUM,ABDIM,CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
```

```
C-------------------------------------------------------------------------
C
C     Purpose
C     -------
C     Computes quotas according to the Revised Management Procedure.
C     Using numerical integration and Brent's algorithm, posterior
C     fractiles are determined, using 4,8,..,1024 point integration rules
C     in turn. Convergence is assumed if the last change is less than
C     accQuota.
C
C
C     Parameters
C     ----------
C      1. OUT_QUOTA - real Output
C See CATCHLIMIT
C
C      2. accQuota - real*8 Input
C See CATCHLIMIT
C
C      3. outDiff - real*8 Output
C       See CATCHLIMIT
C
C      4. npRule - integer Output
C        See CATCHLIMIT
C
C      5. NUM - integer         Input
C See CATCHLIMIT
C
C      6. ABDIM - integer Input
C See CATCHLIMIT
C
C      7. CATCH(NUM) - real array Input
C See CATCHLIMIT
C
C      8. ABEST(ABDIM) - real array Input
C See CATCHLIMIT
C
C      9. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array Input
C See CATCHLIMIT
C
C     10. POP(0:NUM+1) - real*8 array         Workspace
C See CATCHLIMIT
C
C     11. DEVPOP(ABDIM) - real*8 array         Workspace
```

```
C See CATCHLIMIT
C
C     12. AB_YEARS(ABDIM) - integer array Input
C See CATCHLIMIT
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL OUT_QUOTA
      INTEGER npRule, NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM),accQuota,outDiff

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

      REAL PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,PLEVEL
      COMMON /MANPAR/ PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,
     *       PLEVEL

      INTEGER nof_zero,nof_nonz
      REAL*8  dt_lo,dt_hi,logp0_min,logp0_max,fmax,fave
      COMMON/intpar/ nof_zero,nof_nonz,dt_lo,dt_hi,
     *     logp0_min,logp0_max,fmax,fave

      REAL*8 intlow,intupp
      COMMON/fract01/ intlow,intupp

      REAL*8 last_quota,acc,x1,x2,xx2
      INTEGER rulePoint(8),nofRule

C     LOCAL DEFINITIONS
C
      INTEGER i,np,ii1,ii2
      INTEGER PrevfailStatus
      REAL*8 fract,xacc,mu1,mu2,pl1,dt1,dt2,halfInt,r,
     *                                b1,b2,xtry1,xtry2,diff,zbrent
      external fract,halfInt,zbrent
```

```
C-----------------------------------------------------------------------

      data rulePoint/8,16,32,64,128,256,512,1024/,nofRule/8/

C     Stores 0.0 into POP
      DO 10 i=0,NUM
        POP(i) = 0.0
   10 CONTINUE

      logp0_min=-5.0D0
      logp0_max=5.0D1

C     Determine bounds of the search interval
      x1=0.000
      x2=MU_MAX*ABEST(ABDIM)

C     Increase if necessary the upper bound
      ii1 = (ABDIM*(ABDIM-1))/2
      ii2 = (ABDIM*(ABDIM+1))/2
      if (ABDIM.gt.1.and.ABEST(ABDIM-1).gt.ABEST(ABDIM).and.
     *     INFOMATRX(ii1).gt.INFOMATRX(ii2)) then
         xx2 = MU_MAX*ABEST(ABDIM-1)
      else
         xx2 = x2
      endif

      mu1=MU_MIN
      mu2=MU_MAX
      pl1=PLEVEL
      dt1=DT_MIN
      dt2=DT_MAX
      b1=B_MIN
      b2=B_MAX

      acc=accQuota
      xacc=0.25*acc
      if(infoLevel.ge.3)then
         write(IOUT,*) 'calc_quota: required accuracy: ',acc
      endif

C     Start the iteration procedure to determine the PPROB-percentile.

C     Modified November 2000 (previously last_quota=0.0D0)
      last_quota=-1.0D30
```

```
      do i=1,nofRule
         if(infoLevel.ge.2)then
            write(IOUT,*) ' '
         endif
         PrevfailStatus = failStatus
         failStatus=OK
         np=rulePoint(i)
         npRule = np
         if(np.gt.nmax) then
            failStatus = NMAXERR
            return
         endif
C        Setup b-integration.
         call putgauss(np,b1,b2)
C        Setup mu,logp--integration and
C        split integration domain according to DT < > PLEVEL
         call setSplit(rulePoint(i),pl1,mu1,mu2,NUM,CATCH,POP)
         intlow=halfInt(-1,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
         intupp=halfInt(1,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)

C        For peaked integrands intlow+intupp might be zero

         if(intlow+intupp.gt.0.0D0) then
            r=intlow/(intlow+intupp)
         else
            r=-1.0D0
         endif
C
         if(i.eq.1.or.OUT_QUOTA.lt.-1.0D30)then
            xtry1=0.8*x1 + 0.2*x2
            xtry2=0.2*x1 + 0.8*x2
         else
            xtry1=0.95*OUT_QUOTA
            xtry2=1.05*OUT_QUOTA
         endif
         if(r. ge.pprob) then
            OUT_QUOTA=0.0D0
         elseif(r.lt.0.0D0) then
            OUT_QUOTA=-2.0D30
            if (failStatus.eq.OK) failStatus = RIDGEERR
            if(infoLevel.ge.1)then
               write(IOUT,*) 'Approximation could not be computed'
```

```
            endif
         else
            OUT_QUOTA= zbrent(fract,xtry1,xtry2,x1,xx2,xacc,
     *           NUM,ABDIM,CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
            if(OUT_QUOTA.le. -1.0D30)then
               if (failStatus.eq.OK) failStatus = QUOTAERR
               if(infoLevel.ge.1)then
                  write(IOUT,*) 'Solution failed to converge',x1,xx2
               endif
            endif
         endif

         diff=last_quota-OUT_QUOTA
         outDiff = diff

         if(infoLevel.ge.2)then
            write(IOUT,1000)    OUT_QUOTA,diff,rulePoint(i)
         endif
         if(abs(diff).le.acc.and.OUT_QUOTA.ge.-1.0D30 .and.
     *        PrevfailStatus.eq.OK) return
         last_quota=OUT_QUOTA
      enddo
      if (failStatus.eq.OK) failStatus = ACCERR
      if(infoLevel.ge.1)then
         write(IOUT,1100) acc
      endif
      return
 1000 format(1x,'Quota, diff, npoint',
     *                       f12.3,3x,g12.3,2x,2i4)
 1100 format(//,'****** WARNING  ****** ',
     * 'Quota failed to reach required accuracy of',g12.3,//)
      end
C------------------------------------------------------------------------




      REAL*8 FUNCTION lhood(B,NUM,ABDIM,ABEST,INFOMATRX,
     *     POP,DEVPOP,AB_YEARS)
C--------------------------------------------------------------------
C
C     Purpose
C     -------
C     Compute the right-hand side of (4) in [2] for a set of
```

```
C     parameters.
C
C     Parameters
C     ----------
C      1. B - real*8 Input
C On entry: Bias parameter for the abundance estimates
C
C      2. NUM - integer         Input
C         See CATCHLIMIT
C
C      3. ABDIM - integer Input
C See CATCHLIMIT
C
C      4. ABEST(ABDIM) - real array Input
C See CATCHLIMIT
C
C      5. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array Input
C See CATCHLIMIT
C
C      6. POP(0:NUM+1) - real*8 array Input
C On entry: POP(Y) is the population size at year Y;
C              Y = 1,2,...,NUM+1. POP(0) is the population size
C              at year 1.
C
C      7. DEVPOP(ABDIM) - real*8 array Workspace
C              Difference between abundance estimate and population
C              size at years with abundance estimates for various
C              trajectories. This array is indexed in the same way
C              as ABEST.
C
C      8. AB_YEARS(ABDIM) - integer array Input
C      See CATCHLIMIT
C
C     Return value: The likelihood to the power of 1/16
C
C
C-------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8  B
      INTEGER NUM,ABDIM
      REAL ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
```

```
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)
      INTEGER AB_YEARS(ABDIM)

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER initBloop
      COMMON /lhood01/ initBloop

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER I,J,K,ind
      REAL S
      REAL*8 D1,D2,D3,ARG2,DXPP,BETA
C-----------------------------------------------------------------------

      S=1.0D0/1.6D1
      if (initBloop .EQ. 1) then
         initBloop = 0
         do I=1,ABDIM
            ind=AB_YEARS(I)
            if(ind.lt.0) ind=0
            if(POP(ind).le.0.0.and.infoLevel.ge.1)then
               write(IOUT,*) '***** I,ind,POP:',I,ind,POP(ind)
            endif
            DEVPOP(I)=LOG(ABEST(I))-LOG(POP(ind))
            if(infoLevel.ge.6)then
               write(IOUT,*) ' POP, DEVPOP:',
     *              POP(ind),DEVPOP(I)
            endif
         enddo

C     CALCULATES D1, D2 AND D3 defined by (32-34) in [2]

         D1=0.0D0
         D2=0.0D0
         D3=0.0D0
         K=1
         do I=1,ABDIM
            do J=1,I-1
```

```
            D1=D1+2.0D0*INFOMATRX(K)
            D2=D2+2.0D0*INFOMATRX(K)*(DEVPOP(I)+DEVPOP(J))
            D3=D3+2.0D0*INFOMATRX(K)*DEVPOP(I)*DEVPOP(J)
            K=K+1
          enddo
          D1=D1+INFOMATRX(K)
          D2=D2+2.0D0*INFOMATRX(K)*DEVPOP(I)
          D3=D3+INFOMATRX(K)*DEVPOP(I)*DEVPOP(I)
          K=K+1
        enddo
        if(infoLevel.ge.6)then
          write(IOUT,*) ' D1, D2, D3:',D1,D2,D3
        endif
      endif

      BETA=LOG(B)

C     CALCULATES THE LIKELIHOOD FUNCTION

      ARG2= D3 - beta*D2 + beta*beta*D1
      if (0.5D0*ARG2 .GT. 700.) then
        DXPP = 0.0D0
      else
        DXPP=DEXP(-0.5D0*ARG2)
      endif

      if(infoLevel.ge.7)then
        write(IOUT,*) ' Lh (without s) =',DXPP
      endif
      DXPP=DXPP**S
      lhood =  DXPP

      if(infoLevel.ge.7)then
        write(IOUT,*) ' LOG(B):',BETA
        write(IOUT,*) ' lhood = ', lhood
      endif
      RETURN
      END
C------------------------------------------------------------------------




      REAL*8 FUNCTION dens(z1,z2,NUM,ABDIM,
```

```
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Integrate the likelihood w.r.t. the bias parameter.
C     Multiply by the Jacobi determinant of the transformation
C     from (ln(P0),mu,b) to (DT,mu,b). The result is
C     a function of ln(P0) and mu.
C
C     Parameters
C     ----------
C      1. z1 - real*8 Input
C On entry: A value of the parameter mu
C
C      2. z2 - real*8 Input
C On entry: A value of the parameter ln(P0).
C
C      3. NUM - integer         Input
C See CATCHLIMIT
C
C      4. ABDIM - integer Input
C See CATCHLIMIT
C
C      5. CATCH(NUM) - real array         Input
C See CATCHLIMIT
C
C      6. ABEST(ABDIM) - real array         Input
C See CATCHLIMIT
C
C      7. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array Input
C See CATCHLIMIT
C
C      8. POP(0:NUM+1) - real*8 array         Workspace
C See pforw
C
C      9. DEVPOP(ABDIM) - real*8 array         Workspace
C See lhood
C
C     10. AB_YEARS(ABDIM) - integer array Input
C      See CATCHLIMIT
C
C     Return value: Approximation of the innermost integral
C                   of (10) in [2].
```

```
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8    z1,z2
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)

C     GLOBAL DEFINITIONS
C
      INTEGER initBloop
      COMMON /lhood01/ initBloop

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

      REAL*8 fractLT,cutoffDT
      COMMON /dens01/ fractLT,cutoffDT

      INTEGER nof_zero,nof_nonz
      REAL*8  dt_lo,dt_hi,logp0_min,logp0_max,fmax,fave
      COMMON/intpar/ nof_zero,nof_nonz,dt_lo,dt_hi,
     *     logp0_min,logp0_max,fmax,fave

C     LOCAL DEFINITIONS
C
      REAL*8 p0,dt,dpt,d1,z
      REAL*8  PT,LT
      REAL*8 lhood
      external lhood
C-----------------------------------------------------------------------

      if (z2.lt.logp0_min .or. z2.gt.logp0_max) then
         dens=0.0D0
         return
      endif

      call pforw(pt,dpt,z2,z1,NUM,CATCH,POP)

      p0=exp(z2)
      dt=pt/p0
```

```
C     Calculates the absolute value of the Jacobi determinant
      d1=abs(dpt - dt)

C     This (5) in [2]:
      if (dt .GE. cutoffDT ) then
         LT = 3.0D0 * Z1 * (dt-cutoffDT) * PT
      else
         LT = 0.0D0
      endif

      if((dt.gt.dt_lo.and.dt.lt.dt_hi) .and. LT .LT. fractLT) then
         initBloop=1
C        Integrate with respect to the bias parameter.
         call evalgauss(lhood,z,NUM,ABDIM,
     *        ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)

C        Multiply the inner integral with the absolute value of the
C        Jacobi determinant.
         dens = z*d1
         fave=fave+z
         fmax=max(fmax,z)
         nof_nonz=nof_nonz+1

         if(dpt.le.dt.and.infoLevel.ge.7)then
            write(IOUT,*) '**** dens: deriv. PT is less than DT :',
     *           dpt,dt
         endif
      else
         dens=0.0D0
         nof_zero=nof_zero+1
         if(infoLevel.ge.7)then
            write(IOUT,*) 'dens 0 for mu,logp0,dt,lt',z1,z2,dt,lt
         endif
      endif

      RETURN
      END
C-----------------------------------------------------------------------




      SUBROUTINE pforw(pt,dpt,lp0,mu,NUM,CATCH,POP)
C------------------------------------------------------------------------
```

```
C
C     Purpose
C     -------
C     Compute the population size trajectory and the partial derivative
C     of PT w.r.t. ln(P0) (DPT) for a set of parameters.
C
C     In the ordinary case, this is done by using (1) and (9) in [2].
c
C     In the case when POP(I) becomes less than 10**(-30) for some I
C     POP(J) is set to 10**(-30) for all J = I, I+1, ..., T, and
C     DPT is set to zero.
C
C     In the case when POP(0) > 2*10**10, 1-POP(J)/POP(0) may not be
C     accurately computed, and therefore the population size trajectory
C     is computed in two ways. If the results are significantly
C     different, this will be reported through the output value of the
C     parameter IFAIL from the subroutine CATCHLIMIT.
C
C     If 2*PT > 10**30, this will be reported through IFAIL.
C
C     If the Jacobi determinant in (8) in [2] becomes negative while
C     POP(0) is not greater than 10**12, this will be reported through
C     IFAIL.
C
C
C     Parameters
C     ----------
C      1. pt - real*8 Output
C On exit: The population at the year following the last
C               year of historic catch data,
C
C      2. dpt - real*8 Output
C On exit: The partial derivative of pt w.r.t. ln(P0).
C
C      3. lp0 - real*8 Input
C On entry: ln(P0)
C
C      4. mu - real*8           Input
C On entry: Productivity parameter determining the MSY rate
C
C      5. NUM - integer                 Input
C See CATCHLIMIT
C
C      6. CATCH(NUM) - real array Input
```

```
C See CATCHLIMIT
C
C      7. POP(0:NUM+1) - real*8 array Output
C On exit: Population values (NUM+1 corresponds to year T)
C              Y = 1,2,...,NUM+1. POP(0) is the population size
C              at year 1.
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 pt,dpt,lp0,mu
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER J
      REAL*8 R,RATIO
      REAL*8 f1,f2,d1,pt1
C-----------------------------------------------------------------------

      R=1.4184D0*MU
      POP(0)=dexp(lp0)
      POP(1)=POP(0)
      if(POP(0).gt.2.0D10)then
C     POP(0) is large, risk of numerical error.
C     Calculate population trajectory in an alternative way.
         d1=0.0D0
         DO 40 J=1,NUM
            ratio=d1/POP(0)
            d1=-CATCH(J)-2.0D0*R*ratio*POP(j)-R*POP(j)*ratio**2
            POP(J+1)=POP(0)+d1
 40      CONTINUE
         pt1=POP(0)+d1
C     Finished alternative calculation
```

```
      endif

      dpt=1.0D0
      DO J=1,NUM
         POP(J+1)=1.0D-30
      enddo

C     Ordinary case
      DO 50 J=1,NUM
C     This is (1) in [2]:
         RATIO=POP(J)/POP(0)
         POP(J+1)=POP(J)-CATCH(J)
     *                 +R*POP(J)*(1.D0-RATIO)*(1.D0+RATIO)
         if(POP(J+1).lt. 1.0D-30) then
C        Exception case: handle small population size.
C        Avoid zero and negative population size.
            POP(J+1)= 1.0D-30
            goto 200
         endif
C        This is (9) in [2]:
         f1=1.0D0 + R -3.0D0*R*ratio**2
         f2=2.0D0*R*ratio**3
         dpt=f1*dpt + f2
   50 CONTINUE
      pt=POP(NUM+1)
C     Finished ordinary case

      if(POP(0).gt.2.0D10.and.abs((pt-pt1)/pt).gt.1.0D-4)then
C     Significant numerical error.
         if (failStatus.eq.OK) failStatus=PTERR1
         if(infoLevel.ge.1) then
            write(IOUT,*) ' '
            write(IOUT,*) ' pforw: **** WARNING ****'
            write(IOUT,*) ' pforw: **** Numerical error'
            write(IOUT,*) ' pforw: **** pt,pt-pt1:',pt,pt-pt1
         endif
      endif

      if (2.0*pt .gt. 1.0D30) then
C     Large PT.
         if (failStatus.eq.OK) failStatus=PTERR2
         if(infoLevel.ge.1)then
            write(IOUT,*) ' '
            write(IOUT,*) ' pforw: **** WARNING -- Large Pt ****'
```

```
         endif
      endif

      if(dpt.lt.pt/POP(0).and.POP(0).le.1.0D12)then
C     Negative Jacobi determinant. This violates the assumption on
C     which the change of variables in [2] is based.
         if (failStatus.eq.OK) failStatus=JACOBIERR
         if(infoLevel.ge.1)then
            write(IOUT,*) ' pforw: **** WARNING -- Large Pt ****'
            write(IOUT,*) '**** pforw: deriv. PT is less than DT :',
     *           dpt,pt/POP(0),POP(0),mu,pt
         endif
      endif

      RETURN
 200  POP(0)=1.0
      pt=1.0D-30
      dpt=0.0
      return
      END
C-----------------------------------------------------------------------



      SUBROUTINE grule(n,x,w)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     This subroutine computes the [(n+1)/2] nonnegative abscissas
C     x(i) and corresponding weights w(i) of the n-point Gauss-Legendre
C     integration rule, normalized to the interval [-1,1]. The abscissas
C     appear in descending order.
C
C     Parameters
C     ----------
C      1. n - integer Input
C On entry: The number of points used in integration rule
C
C      2. x - real*8 array Output
C On exit: The points used in the integration rule
C
C      3. w - real*8 array Output
```

```
C On exit: The weights used in the integration rule
C
C     Reference
C     ---------
C     This is the routine "GRULE" at page 369 in [3].
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 x(*),w(*)
      INTEGER n

C     LOCAL DEFINITIONS
C
      REAL*8 pkm1,pk,t1,pkp1,den,d1,dpn,d2pn,d3pn,d4pn,
     *                          u,v,h,p,dp,fx,e1,t,x0
      INTEGER m,i,k
C-----------------------------------------------------------------------

      m=(n+1)/2
      e1=n*(n+1)
      do i=1,m
         t=(4*i-1)*3.1415926536D0/(4*n+2)
         x0=(1.0D0-(1.0D0-1.0D0/n)/(8.0D0*n*n))*cos(t)
         pkm1=1.0D0
         pk=x0
         do k=2,n
            t1=x0*pk
            pkp1=t1-pkm1-(t1-pkm1)/k+t1
            pkm1=pk
            pk=pkp1
         enddo
         den=1.0D0-x0*x0
         d1=n*(pkm1-x0*pk)
         dpn=d1/den
         d2pn=(2.0D0*x0*dpn-e1*pk)/den
         d3pn=(4.0D0*x0*d2pn+(2.0D0-e1)*dpn)/den
         d4pn=(6.0D0*x0*d3pn+(6.0D0-e1)*d2pn)/den
         u=pk/dpn
         v=d2pn/dpn
         h=-u*(1.0D0+0.5D0*u*(v+u*(v*v-u*d3pn/(3.0D0*dpn))))
         p=pk+h*(dpn+0.5D0*h*(d2pn+
```

```
     *                        h/3.0D0*(d3pn+0.25D0*h*d4pn)))
            dp=dpn+h*(d2pn+0.5D0*h*(d3pn+h*d4pn/3.0D0))
            h=h-p/dp
            x(i)=x0+h
            fx=d1-h*e1*(pk+0.5D0*h*(dpn+h/3.0D0*(d2pn+
     *                      0.25D0*h*(d3pn+0.2D0*h*d4pn))))
            w(i)=2.0D0*(1.0D0-x(i)*x(i))/(fx*fx)
      enddo
      if (m+m.gt.n) x(m)=0.0D0
      return
      end
C-----------------------------------------------------------------------




      SUBROUTINE putgauss(n,a,b)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Set up the coefficients for a n-point Gauss-Legendre integration
C     rule on [a,b]. The result is stored in xg and wg.
C
C     Parameters
C     ----------
C      1. n - integer Input
C On entry: The number of points used in integration
C Constraint: n is even
C
C      2. a - real*8 Input
C On entry: The left endpoint of the integration interval
C
C      3. b - real*8 Input
C On entry: The right endpoint of the integration interval
C
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER n
      REAL*8 a,b
```

```
C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

C     LOCAL DEFINITIONS
C
      INTEGER i,m
      REAL*8 c,d
      REAL*8 x0(nmax),w0(nmax)
C-----------------------------------------------------------------------

      ngauss=n
      call grule(n,x0,w0)
      m=(n+1)/2
      c=(a+b)/2
      d=(b-a)/2
      do i=1,m
         xg(i)=c-d*x0(i)
         xg(m+i)=c+d*x0(m-i+1)
         wg(i)=d*w0(i)
         wg(m+i)=d*w0(m-i+1)
      enddo
      return
      end
C-----------------------------------------------------------------------



      SUBROUTINE evalgauss(func,value,NUM,ABDIM,
     *     ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Approximate a one-dimensional integral of a function using
C     the Gauss-Legendre integration rule
C
C     Parameters
C     ----------
C      1. func - real*8 function External
C      The function to be integrated.
C Its specification is:
```

```
C REAL*8 FUNCTION  func(B,NUM,ABDIM,A,V,POP,DEVPOP,AB_YEARS)
C REAL*8  B
C       INTEGER NUM,ABDIM
C       REAL ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
C       REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)
C       INTEGER AB_YEARS(ABDIM)
C Parameters of func: See lhood
C
C      2. value - real*8 Output
C On exit: The (approximated) value of the integral
C
C      3. NUM - integer         Input
C See CATCHLIMIT
C
C      4. ABDIM - integer Input
C See CATCHLIMIT
C
C      5. ABEST(ABDIM) - real array                     Input
C See CATCHLIMIT
C
C      6. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array     Input
C See CATCHLIMIT
C
C      7. POP(0:NUM+1) - real*8 array                   Input
C See lhood
C
C      8. DEVPOP(ABDIM) - real*8 array                  Workspace
C See lhood
C
C      9. AB_YEARS(ABDIM) - integer array      Input
C See CATCHLIMIT
C
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 func,value
      external func
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)
```

```
C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

C     LOCAL DEFINITIONS
C
      INTEGER i
C-----------------------------------------------------------------------

      value=0.0D0
      do i=1,ngauss
         value=value+wg(i)*func(xg(i),
     *        NUM,ABDIM,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
      enddo

      return
      end
C-----------------------------------------------------------------------



      SUBROUTINE prodgauss(n2,a1,b1,NUM,CATCH,POP)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Set up integration w.r.t. mu and ln(P0).
C     The results are stored in xg1, wg1, xg12, and wg12.
C
C     Parameters
C     ----------
C      1. n2 - integer Input
C On entry: The number of points used in the integration
C               w.r.t. ln(P0)
C
C      2. a1 - real*8 Input
C On entry: Left endpoint of the interval of integration
C               w.r.t. mu
C
C      3. b1 - real*8 Input
C On entry: Right endpoint of the interval of integration
C               w.r.t. mu
C
```

```
C      4. NUM - integer         Input
C         See CATCHLIMIT
C
C      5. CATCH(NUM) - real array         Input
C         See CATCHLIMIT
C
C      6. POP(0:NUM+1) - real*8 array         Workspace
C         See CATCHLIMIT
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER n2
      REAL*8 a1,b1
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      REAL*8 fractLT,cutoffDT
      COMMON /dens01/ fractLT,cutoffDT

      REAL*8 xmu,lt,plvl1
      COMMON /intLvl01/ xmu,lt,plvl1

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER n1,i,j,m2
      REAL*8 x0(nmax),w0(nmax),c,a2,b2,d,
     *     ap1,ap2,bp1,bp2,del,xbrent,intLevel,getSplit
      external xbrent,intLevel,getSplit
C-----------------------------------------------------------------------

      n1=ng1
      ng2=n2
      call grule(n2,x0,w0)
```

```
      m2=(n2+1)/2
      del=0.2D0
      ap1=0.0D0
      ap2=1.0D0
      bp1=10.0D0
      bp2=20.0D0
      do i=1,n1
         a2=getSplit(xg1(i))

         plvl1=cutoffDT
         lt=fractLT
         if(lt.lt.0.0D0)then
            b2=-1.0D30
            if(failStatus.eq.OK) failStatus = LTERR
            if(infoLevel.ge.1) write(IOUT,*) 'LT is negative'
         else
            xmu=xg1(i)
            b2=xbrent(intLevel,bp1,bp2,-5.0D0,5.0D1,1.0D-12,
     *           NUM,CATCH,POP)
         endif

         if(min(a2,b2).le.-1.0D30)then
            if (failStatus.eq.OK) failStatus = PRODGAUSSERR
            if(infoLevel.ge.1)then
               write(IOUT,*) 'prodGauss: could not find limits:',xg1(i)
            endif
            a2=ap1
            b2=bp2
         endif
         if(infoLevel.ge.4.and.(i.eq.1.or.i.eq.n1))then
            write(IOUT,1000) a2,b2,xg1(i)
         endif
         c=(a2+b2)/2
         d=(b2-a2)/2
         ap1=a2-del
         ap2=a2+del
         bp1=b2-del
         bp2=b2+del
         do j=1,m2
            xg2(i,j)=c-d*x0(j)
            xg2(i,m2+j)=c+d*x0(m2-j+1)
            wg12(i,j)=d*w0(j)*wg1(i)
            wg12(i,m2+j)=d*w0(m2-j+1)*wg1(i)
         enddo
```

```
      enddo

      return
 1000 format(1x,'prodgauss,limits:',2f8.3,2x,f10.5)
      end
C-----------------------------------------------------------------------------



      SUBROUTINE halfgauss(n2,a1,b1,para,NUM,CATCH,POP)
C-----------------------------------------------------------------------------
C
C     Purpose
C     -------
C     Set up integration w.r.t. mu and ln(P0).
C     The results are stored in xg1, wg1, xg12, and wg12.
C
C     Rule is ng1(or n1) x 2*n2
C
C
C     Parameters
C     ----------
C      1. n2 - integer Input
C On entry: The number of points used in the integration
C              w.r.t. ln(P0)
C
C      2. a1 - real*8 Input
C On entry: Left endpoint of the interval of integration
C              w.r.t. mu
C
C      3. b1 - real*8 Input
C On entry: Right endpoint of the interval of integration
C              w.r.t. mu
C
C      4. para - real*8 Input
C On entry: WRITE MORE!
C
C      5. NUM - integer         Input
C         See CATCHLIMIT
C
C      6. CATCH(NUM) - real array         Input
C         See CATCHLIMIT
C
```

```
C      7. POP(0:NUM+1) - real*8         Workspace
C         See CATCHLIMIT
C
C------------------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER n2
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)
      REAL*8 a1,b1,para

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER nof_zero,nof_nonz
      REAL*8  dt_lo,dt_hi,logp0_min,logp0_max,fmax,fave
      COMMON/intpar/ nof_zero,nof_nonz,dt_lo,dt_hi,
     *     logp0_min,logp0_max,fmax,fave

      REAL*8 mu01,ldt0
      COMMON /lptoldt01/ mu01,ldt0

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER n1,m2,m3,i,j
      REAL*8 x0(nmax),w0(nmax),c,a2,b2,e(nmax),f(nmax),d,
     *                         bp1,bp2,del
      REAL*8 xbrent,logptoldt,getSplit
      external xbrent,logptoldt,getSplit
C------------------------------------------------------------------------------

      n1=ng1
      ng2=n2+n2

      call grule(n2,x0,w0)
      m2=(n2+1)/2
      del=1.0D0
```

```
      bp1=0.0D0
      bp2=bp1+del
      do i=1,n1
         a2=getSplit(xg1(i))
         e(i)=a2
         ldt0=log(para)
         mu01=xg1(i)
         b2=xbrent(logptoldt,bp1,bp2,logp0_min,logp0_max,
     *     1.0D-12,NUM,CATCH,POP)
         if(b2.le.-1.0D30) b2=-1.1D-30
         if(b2.le.-1.0D30)then
            if(failStatus.eq.OK) failStatus = HALFGAUSSERR
            if(infoLevel.ge.1)then
               write(IOUT,*) 'halfGauss: could not find limit at ',
     *              xg1(i)
            endif
         endif
         f(i)=b2
         if(infoLevel.ge.4 .and. (i.eq.n1.or.i.eq.1))then
            write(IOUT,1000) e(i),f(i),xg1(i)
         endif
         c=(a2+b2)/2
         d=(b2-a2)/2
         bp1=b2-del
         bp2=b2+del
         do j=1,m2
            xg2(i,j)=c-d*x0(j)
            xg2(i,m2+j)=c+d*x0(m2-j+1)
            wg12(i,j)=d*w0(j)*wg1(i)
            wg12(i,m2+j)=d*w0(m2-j+1)*wg1(i)
         enddo
      enddo

C          copies the coefficients untransformed first to [-1,1]
      m3=m2
      do i=1,n1
         c=0.0D0
         d=1.0D0
         do j=1,m2
            xg2(i,n2+j)=c-d*x0(j)
            xg2(i,n2+m3+j)=c+d*x0(m3-j+1)
            wg12(i,n2+j)=d*w0(j)*wg1(i)
            wg12(i,n2+m3+j)=d*w0(m3-j+1)*wg1(i)
         enddo
```

```
      enddo

C          transforms the coefficients from [-1,1] to the 2. interval
      do i=1,n1
         a2=f(i)
         b2=f(i)-e(i)
         do j=1,n2
            wg12(i,n2+j)=b2*2.0D0*wg12(i,n2+j)
     *                               /(1.0D0-xg2(i,n2+j))**2
            xg2(i,n2+j)=a2+b2*(2.0D0/(1.0D0-xg2(i,n2+j))-1.0D0)
         enddo
      enddo

      return
 1000 format(1x,'halfgauss, limits:',2f8.3,2x,f10.5)
      end
C-----------------------------------------------------------------------------




      SUBROUTINE evalpgauss(func,value,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C-----------------------------------------------------------------------------
C
C     Purpose
C     -------
C     Approximate a two-dimensional integral of a function by using
C     iterated itegration and Gauss-Legendre rules to evaluate the
C     iterated integrals.
C
C     Parameters
C     ----------
C      1. func - real*8 function External
C      The function to be integrated.
C Its specification is:
C       REAL*8 FUNCTION func(z1,z2,NUM,ABDIM,
C      *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C       REAL*8    z1,z2
C       INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
C       REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
C       REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)
C Parameters of func: See dens
C
```

```
C      2. value - real*8 Output
C On exit: The (approximated) value of the integral
C
C      3. NUM - integer         Input
C         See CATCHLIMIT
C
C      4. ABDIM - integer Input
C         See CATCHLIMIT
C
C      5. CATCH(NUM) - real array         Input
C         See CATCHLIMIT
C
C      6. ABEST(ABDIM) - real array         Input
C         See CATCHLIMIT
C
C      7. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array Input
C         See CATCHLIMIT
C
C      8. POP(0:NUM+1) - real*8 array         Workspace
C         See CATCHLIMIT
C
C      9. DEVPOP(ABDIM) - real*8 array                 Workspace
C         See CATCHLIMIT
C
C     10. AB_YEARS(ABDIM)  - integer array Input
C         See CATCHLIMIT
C
C-----------------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 func,value
      external func
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

C     LOCAL DEFINITIONS
C
```

```
      INTEGER i,j
C-----------------------------------------------------------------------------

      value=0.0D0
      do i=1,ng1
         do j=1,ng2
            value=value+wg12(i,j)*func(xg1(i),xg2(i,j),
     *           NUM,ABDIM,
     *           CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
         enddo
      enddo

      return
      end
C-----------------------------------------------------------------------




      REAL*8 FUNCTION logptoldt(lp,NUM,CATCH,POP)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Compute ln(PT)-ln(P0)-ln(DT)
C
C     Parameters
C     ----------
C      1. lp - real*8 Input
C On entry: ln(P0)
C
C      2. NUM - integer         Input
C See CATCHLIMIT
C
C      3. CATCH(NUM) - real array         Input
C     See CATCHLIMIT
C
C      4. POP(0:NUM+1) - real*8 array         Workspace
C     See pforw
C
C
C     Return value: ln(PT)-ln(P0)-ln(DT)
C
C-----------------------------------------------------------------------
```

```
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 lp
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)

C     GLOBAL DEFINITIONS
C
      REAL*8 mu01,ldt0
      COMMON /lptoldt01/ mu01,ldt0

C     LOCAL DEFINITIONS
C
      REAL*8 pt,dpt
C-----------------------------------------------------------------------

      call pforw(pt,dpt,lp,mu01,NUM,CATCH,POP)

      logptoldt= log(pt)-lp-ldt0

      return
      end
C-----------------------------------------------------------------------



      REAL*8 FUNCTION intLevel(x1,NUM,CATCH,POP)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Determine the internal catch limit as a function of ln(P0),
C     see (5) in [2],
C     mu is considered as a fixed parameter.
C     Subtract a fixed quantity lt from the result.
C
C     Parameters
C     ----------
C      1. x1 - real*8 Input
C On entry: ln(P0)
```

```
C
C      2. NUM - integer         Input
C See CATCHLIMIT
C
C      3. CATCH(NUM) - real array         Input
C See CATCHLIMIT
C
C      4. POP(0:NUM+1) - real*8 array         Workspace
C See pforw
C
C     Return value: The internal catch limit - lt
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 x1
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)

C     GLOBAL DEFINITIONS
C
      REAL*8 xmu,lt,plvl1
      COMMON /intLvl01/ xmu,lt,plvl1

C     LOCAL DEFINITIONS
C
      REAL*8 pt1,dpt,dt
C-----------------------------------------------------------------------

      call pforw(pt1,dpt,x1,xmu,NUM,CATCH,POP)
      dt=pt1*dexp(-x1)
C     This is (5) in [2]:
      intLevel=3.0D0*xmu*pt1*max(0.0D0,dt-plvl1)-lt
      return
      end
C-----------------------------------------------------------------------



      REAL*8 FUNCTION xbrent(func,x1,x2,xb1,xb2,tol,NUM,CATCH,POP)
```

```
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Solve the equation func(x)=0 using Brent's method.
C
C     Parameters
C     ----------
C      1. func - real*8 function External
C Its specification is:
C       REAL*8 FUNCTION func(lp,NUM,CATCH,POP)
C       REAL*8 lp
C       INTEGER NUM
C       REAL CATCH(NUM)
C       REAL*8 POP(0:NUM+1)
C Parameters of func: See logptoldt or intLevel
C
C      2. x1 - real*8 Input
C On entry: Left endpoint of initial search interval
C
C      3. x2 - real*8 Input
C On entry: Right endpint of initial search interval
C
C      4. xb1 - real*8 Input
C On entry: Lower bound for the solution
C
C      5. xb2 - real*8 Input
C On entry: Upper bound for the solution
C
C      6. tol - real*8 Input
C On entry: Parameter determining the accuracy of the solution
C
C      7. NUM - integer         Input
C See CATCHLIMIT
C
C      8. CATCH(NUM) - real array         Input
C     See CATCHLIMIT
C
C      9. POP(0:NUM+1) - real*8 array         Workspace
C     See CATCHLIMIT
C
C     Return value: The zero of the function 'func'
C
C-----------------------------------------------------------------------
```

```
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 func,x1,x2,xb1,xb2,tol
      external func
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)

C     LOCAL DEFINITIONS
C
      INTEGER itmax
      REAL*8 eps,xmiss
      parameter(itmax=10000,eps=1.0D-12,xmiss=-1.1D30)
      REAL*8 a,b,fa,fb,fc,tol1,c,d,e,xm,p,q,r,s
      INTEGER iter
C-----------------------------------------------------------------------

      a=x1
      b=x2
      fa=func(a,NUM,CATCH,POP)
      fb=func(b,NUM,CATCH,POP)
      if(fa*fb.gt.0.0D0)then
         a=xb1
         fa=func(a,NUM,CATCH,POP)
         if(fa*fb.gt.0.0D0)then
            b=xb2
            fb=func(b,NUM,CATCH,POP)
         endif
      endif
      if(fa*fb.gt.0.0D0) goto 100
      fc=fb
      do iter=1,itmax
         if(fb*fc.gt.0.0D0)then
            c=a
            fc=fa
            d=b-a
            e=d
         endif
         if(abs(fc).lt.abs(fb))then
            a=b
            b=c
            c=a
```

```
            fa=fb
            fb=fc
            fc=fa
         endif
         tol1=2.0D0*eps*abs(b) + 0.5D0*tol
         xm=0.5D0*(c-b)
         if(abs(xm).le.tol1 .or. fb.eq.0.0D0)then
            xbrent=b
            return
         endif
         if(abs(e).ge.tol1 .and. abs(fa).gt.abs(fb)) then
            s=fb/fa
            if(a.eq.c)then
               p=2.0D0*xm*s
               q=1.0D0-s
            else
               q=fa/fc
               r=fb/fc
               p=s*(2.0D0*xm*q*(q-r) - (b-a)*(r-1.0D0))
               q=(q-1.0D0)*(r-1.0D0)*(s-1.0D0)
            endif
            if(p.gt.0.0D0) q=-q
            p=abs(p)
            if(2.0D0*p.lt.min(3.0D0*xm*q-abs(tol1*q),abs(e*q)))then
               e=d
               d=p/q
            else
               d=xm
               e=d
            endif
         else
            d=xm
            e=d
         endif
         a=b
         fa=fb
         if(abs(d).gt.tol1)then
            b=b+d
         else
            b=b+sign(tol1,xm)
         endif
         fb=func(b,NUM,CATCH,POP)
      enddo
 100  xbrent=xmiss
```

```
      return
      end
C-----------------------------------------------------------------------




      REAL*8 FUNCTION zbrent(func,x1,x2,xb1,xb2,tol,
     *     NUM,ABDIM,CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C-----------------------------------------------------------------------
C
C     Purpose
C     -------
C     Solve the equation func(x)=0 using Brent's method.
C     Except for some additional parameters, this function is equal to
C     the function 'xbrent'. The two copies are needed in order to
C     avoid recursion.
C
C     Parameters
C     ----------
C      1. func - real*8 function External
C Its specification is:
C       REAL*8 FUNCTION func(x,NUM,ABDIM,
C      *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C       REAL*8 x
C       INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
C       REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
C       REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)
C Parameters of func: See fract
C
C      2. x1 - real*8 Input
C On entry: Left endpoint of initial search interval
C
C      3. x2 - real*8 Input
C On entry: Right endpint of initial search interval
C
C      4. xb1 - real*8 Input
C On entry: Lower bound for the solution
C
C      5. xb2 - real*8 Input
C On entry: Upper bound for the solution
C
C      6. tol - real*8 Input
C On entry: Parameter determining the accuracy of the solution
```

```
C
C      7. NUM - integer         Input
C See CATCHLIMIT
C
C      8. ABDIM - integer Input
C See CATCHLIMIT
C
C      9. CATCH(NUM) - real array         Input
C See CATCHLIMIT
C
C     10. ABEST(ABDIM) - real array         Input
C See CATCHLIMIT
C
C     11. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array     Input
C See CATCHLIMIT
C
C     12. POP(0:NUM+1) - real*8 array         Workspace
C See CATCHLIMIT
C
C     13. DEVPOP(ABDIM) - real*8 array         Workspace
C See CATCHLIMIT
C
C     14. AB_YEARS(ABDIM) - integer array Input
C      See CATCHLIMIT
C
C     Return value: The zero of the function 'func'
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 func,x1,x2,xb1,xb2,tol
      external func
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)

C     LOCAL DEFINITIONS
C
      INTEGER itmax
      REAL*8 eps,xmiss
      parameter(itmax=10000,eps=1.0D-12,xmiss=-1.1D30)
      REAL*8 a,b,fa,fb,fc,tol1,c,d,e,xm,p,q,r,s
```

```
      INTEGER iter
C-----------------------------------------------------------------------

      a=x1
      b=x2
      fa=func(a,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
      fb=func(b,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
      if(fa*fb.gt.0.0D0)then
         a=xb1
         fa=func(a,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
         if(fa*fb.gt.0.0D0)then
            b=xb2
            fb=func(b,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
         endif
      endif
      if(fa*fb.gt.0.0D0) goto 100
      fc=fb
      do iter=1,itmax
         if(fb*fc.gt.0.0D0)then
            c=a
            fc=fa
            d=b-a
            e=d
         endif
         if(abs(fc).lt.abs(fb))then
            a=b
            b=c
            c=a
            fa=fb
            fb=fc
            fc=fa
         endif
         tol1=2.0D0*eps*abs(b) + 0.5D0*tol
         xm=0.5D0*(c-b)
         if(abs(xm).le.tol1 .or. fb.eq.0.0D0)then
            zbrent=b
            return
         endif
         if(abs(e).ge.tol1 .and. abs(fa).gt.abs(fb)) then
            s=fb/fa
```

```
          if(a.eq.c)then
             p=2.0D0*xm*s
             q=1.0D0-s
          else
             q=fa/fc
             r=fb/fc
             p=s*(2.0D0*xm*q*(q-r) - (b-a)*(r-1.0D0))
             q=(q-1.0D0)*(r-1.0D0)*(s-1.0D0)
          endif
          if(p.gt.0.0D0) q=-q
          p=abs(p)
          if(2.0D0*p.lt.min(3.0D0*xm*q-abs(tol1*q),abs(e*q)))then
             e=d
             d=p/q
          else
             d=xm
             e=d
          endif
       else
          d=xm
          e=d
       endif
       a=b
       fa=fb
       if(abs(d).gt.tol1)then
          b=b+d
       else
          b=b+sign(tol1,xm)
       endif
       fb=func(b,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
      enddo
 100  zbrent=xmiss
      return
      end
C-----------------------------------------------------------------------------




      REAL*8 FUNCTION getSplit(mu)
C------------------------------------------------------------------------
C
```

```
C     Purpose
C     -------
C     Get the value of ln(P0) such that DT = the internal protection level
C     for a given mu.
C
C     Parameters
C     ----------
C      1. mu - real*8 Input
C On entry: mu
C
C     Return value: The appropriate value of ln(P0)
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 mu

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

C     LOCAL DEFINITIONS
C
      INTEGER i
C-----------------------------------------------------------------------

      do i=1,ng1
         if(abs(mu-xg1(i)).lt.(1.0D0+mu)*1.0D-20)then
            getSplit=intSplit(i)
            return
         endif
      enddo
      getSplit=0.0D0
      return
      end
C-----------------------------------------------------------------------




      SUBROUTINE setSplit(npoint,dtSplit,a1,b1,NUM,CATCH,POP)
C-----------------------------------------------------------------------
```

```
C
C     Purpose
C     -------
C     Find the abscissas and weights for the mu-integral.
C     Determine and store the value of ln(P0) such that
C     DT = the internal protection level for each mu used as
C     abscissa in the integration rule.
C
C     Parameters
C     ----------
C      1. npoint - integer Input
C On entry: The number of points in the integration rule
C
C      2. dtSplit - real*8 Input
C On entry: The internal protection level
C
C      3. a1 - real*8 Input
C On entry: Left endpoint of integration interval w.r.t. mu
C
C      4. b1 - real*8 Input
C On entry: Right endpoint of integration interval w.r.t. mu
C
C      5. NUM - integer         Input
C  See CATCHLIMIT
C
C      6. CATCH(NUM) - real array         Input
C See CATCHLIMIT
C
C      7. POP(0:NUM+1) - real*8 array         Workspace
C See CATCHLIMIT
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER npoint
      REAL*8 dtSplit,a1,b1
      INTEGER NUM
      REAL CATCH(NUM)
      REAL*8 POP(0:NUM+1)

C     GLOBAL DEFINITIONS
C
```

```
      include 'xrmpSub_inc.f'

      INTEGER nof_zero,nof_nonz
      REAL*8  dt_lo,dt_hi,logp0_min,logp0_max,fmax,fave
      COMMON/intpar/ nof_zero,nof_nonz,dt_lo,dt_hi,
     *     logp0_min,logp0_max,fmax,fave

      REAL*8 mu01,ldt0
      COMMON /lptoldt01/ mu01,ldt0

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      INTEGER n1
      REAL*8 x0(nmax),w0(nmax),c,a2,d,ap1,ap2,del
      INTEGER m1,i
      REAL*8 xbrent,logptoldt
      external xbrent,logptoldt
C-----------------------------------------------------------------------

      n1=npoint
      ng1=n1
      call grule(n1,x0,w0)
      m1=(ng1+1)/2
      c=(a1+b1)/2
      d=(b1-a1)/2
      do i=1,m1
         xg1(i)=c-d*x0(i)
         xg1(m1+i)=c+d*x0(m1-i+1)
         wg1(i)=d*w0(i)
         wg1(m1+i)=d*w0(m1-i+1)
      enddo

      del=1.0D0
      ap1=0.0D0
      ap2=1.0D0
      do i=1,ng1
         ldt0=log(dtSplit)
         mu01=xg1(i)
         a2=xbrent(logptoldt,ap1,ap2,logp0_min,logp0_max,
     *     1.0D-12,NUM,CATCH,POP)
         if(a2.le.-1.0D30) a2=-1.1D-30
```

```
         if(a2.le.-1.0D30)then
            if (failStatus.eq.OK) failStatus = SETSPLITERR
            if(infoLevel.ge.1)then
               write(IOUT,*) 'could not set split',dtSplit
            endif
         endif
         intSplit(i)=a2
         if(infoLevel.ge.4.and.(i.eq.1.or.i.eq.ng1))then
            write(IOUT,1000) i,intSplit(i),xg1(i)
         endif
         ap1=a2-del
         ap2=a2+del
      enddo

      return
 1000 format(1x,' split at:',i4,2f10.4)
      end
C------------------------------------------------------------------------




      REAL*8 FUNCTION halfInt(imode,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C-------------------------------------------------------------------------
C
C     Purpose
C     -------
C     Calculate the semi-infinite integral of the likelihood
C     (wrt. log(P0) )
C
C     Parameters
C     ----------
C      1. imode - integer Input
C On entry: If imode < 0, the integral is from -infinity
C              to split (ln(P0) corresponding to the internal
C              protection level). Otherwise, the integral is from split
C              to infinity.
C
C      2. NUM - integer        Input
C See CATCHLIMIT
C
C      3. ABDIM - integer Input
C See CATCHLIMIT
```

```
C
C      4. CATCH(NUM) - real array         Input
C See CATCHLIMIT
C
C      5. ABEST(ABDIM) - real array         Input
C See CATCHLIMIT
C
C      6. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array     Input
C See CATCHLIMIT
C
C      7. POP(0:NUM+1) - real*8 array         Workspace
C See CATCHLIMIT
C
C      8. DEVPOP(ABDIM) - real*8 array         Workspace
C See CATCHLIMIT
C
C      9. AB_YEARS(ABDIM) - integer array Input
C       See CATCHLIMIT
C
C     Return value: The (approximate) value of the integral
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      INTEGER imode
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      REAL PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,PLEVEL
      COMMON /MANPAR/ PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,
     *       PLEVEL

      REAL*8 fractLT,cutoffDT
      COMMON /dens01/fractLT,cutoffDT

      INTEGER nof_zero,nof_nonz
      REAL*8  dt_lo,dt_hi,logp0_min,logp0_max,fmax,fave
```

```
      COMMON/intpar/ nof_zero,nof_nonz,dt_lo,dt_hi,
     *     logp0_min,logp0_max,fmax,fave

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

C     LOCAL DEFINITIONS
C
      REAL*8 dens,plow,phigh,getSplit,a1,b1,finest,dt1
      EXTERNAL dens,plow,phigh,getSplit
      INTEGER n1,n2,n3
C-------------------------------------------------------------------------

      fmax=-100.0
      fave=0.0D0
      nof_zero=0
      nof_nonz=0
      n1=ng1
      n2=ng1
      n3=ng1
      a1=mu_min
      b1=mu_max
      fractLT=1.0D30
      cutoffDT=PLEVEL

      if(imode.lt.0)then
C        integral from -infty to split
         dt_lo=DT_MIN
         dt_hi=PLEVEL
         dt1=0.8*dt_lo + 0.2*dt_hi
         call halfgauss(n2,a1,b1,dt1,NUM,CATCH,POP)
         call evalpgauss(dens,finest,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
         finest=-finest
      else
C        integral from split to +infty
         dt_lo=PLEVEL
         dt_hi=DT_MAX
         dt1=0.8*dt_hi + 0.2*dt_lo
         call halfgauss(n2,a1,b1,dt1,NUM,CATCH,POP)
         call evalpgauss(dens,finest,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
      endif
```

```
      halfInt=finest
      fave=fave/(nof_zero+nof_nonz)
      if(infoLevel.ge.3)then
         write(IOUT,*)
     *           'halfInt: integral, fmax, fave,nof_zero,nof_nonz ',
     *                          finest,fmax,fave,nof_zero,nof_nonz
      endif
      return
      end
C------------------------------------------------------------------------



      REAL*8 FUNCTION fract(x,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)
C------------------------------------------------------------------------
C
C     Purpose
C     -------
C     Calculate the cumulative probability of the internal catch limit at x.
C     Subtract the probability level (PPROB) from the result.
C
C     Parameters
C     ----------
C      1. x - real*8 Input
C On entry: A given internal catch limit
C
C      2. NUM - integer         Input
C See CATCHLIMIT
C
C      3. ABDIM - integer Input
C See CATCHLIMIT
C
C      4. CATCH(NUM) - real array         Input
C See CATCHLIMIT
C
C      5. ABEST(ABDIM) - real array         Input
C See CATCHLIMIT
C
C      6. INFOMATRX(ABDIM*(ABDIM+1)/2) - real array     Input
C See CATCHLIMIT
C
C      7. POP(0:NUM+1) - real*8 array         Workspace
```

```
C See CATCHLIMIT
C
C       8. DEVPOP(ABDIM) - real*8 array         Workspace
C See CATHCLIMIT
C
C       9. AB_YEARS(ABDIM) - integer array Input
C       See CATCHLIMIT
C
C      Return value: cumulative probability - PPROB
C
C
C-----------------------------------------------------------------------
      IMPLICIT NONE

C     PARAMETERS
C
      REAL*8 x
      INTEGER NUM,ABDIM,AB_YEARS(ABDIM)
      REAL CATCH(NUM),ABEST(ABDIM),INFOMATRX(ABDIM*(ABDIM+1)/2)
      REAL*8 POP(0:NUM+1),DEVPOP(ABDIM)

C     GLOBAL DEFINITIONS
C
      include 'xrmpSub_inc.f'

      INTEGER infoLevel,failStatus,IOUT
      COMMON/infopar/ infoLevel,failStatus,IOUT

      REAL*8 intlow,intupp
      COMMON/fract01/ intlow,intupp

      REAL PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,PLEVEL
      COMMON /MANPAR/ PPROB,MU_MIN,MU_MAX,DT_MIN,DT_MAX,B_MIN,B_MAX,
     *       PLEVEL

      REAL*8 fractLT,cutoffDT
      COMMON /dens01/ fractLT,cutoffDT

C     LOCAL DEFINITIONS
C
      REAL*8 ratio,intdelta,a,b,dens
      external dens
      INTEGER n1
C-----------------------------------------------------------------------
```

```
      n1=ng1
      a=mu_min
      b=mu_max
      cutoffDT=PLEVEL
      fractLT=x
      call prodgauss(n1,a,b,NUM,CATCH,POP)
      call evalpgauss(dens,intdelta,NUM,ABDIM,
     *     CATCH,ABEST,INFOMATRX,POP,DEVPOP,AB_YEARS)

      ratio= (intlow+intdelta)/(intlow+intupp)

      if(infoLevel.ge.3)then
         write(IOUT,*) 'zbrent solver: fract,prob:',x,ratio
      endif

      fract=ratio - pprob
      return
      end
C-----------------------------------------------------------------------
```

```
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
C     xrmpSub_inc.f
C
C     Include file for xrmpSub.f
C     Norwegian Computing Center, december 1992, Jon Helgeland
C     Modified, April 1999, Ragnar Bang Huseby
C
C
C     Error message constants
C     -----------------------
C     On exit of the subroutine CATCHLIMIT, IFAIL is equal to one of
C     these constants.
C
      integer OK,NUMERR,ABDIMERR,CATCHERR,AERR,VERR,
     *     ABYEARERR,NMAXERR,
```

```
     *      PPROBERR,MUERR,DTERR,BERR,PLEVELERR,ACCQUOTAERR,PTERR1,
     *      PTERR2,JACOBIERR,SETSPLITERR,HALFGAUSSERR,PRODGAUSSERR,LTERR,
     *      RIDGEERR,QUOTAERR,ACCERR,IFAILERR
      parameter(OK=0,NUMERR=2,ABDIMERR=3,CATCHERR=4,AERR=5,VERR=6,
     *      ABYEARERR=7,PPROBERR=8,
     *      MUERR=9,DTERR=10,BERR=11,PLEVELERR=12,ACCQUOTAERR=13,
     *      NMAXERR=14,PTERR1=15,PTERR2=16,JACOBIERR=17,SETSPLITERR=18,
     *      HALFGAUSSERR=19,PRODGAUSSERR=20,
     *      LTERR=21,RIDGEERR=22,QUOTAERR=23,ACCERR=24,IFAILERR=-2)


C     Variables used by Gauss-Legendre integration rules
C     --------------------------------------------------
C
C     nmax specifies the size of the arrays xg, wg, xg1, xg2, wg1, wg12,
C     and intSplit. nmax is a constant. nmax should be at least as large
C     as the maximum number of abscissas used in the integration rules.
C
C     xg and wg contain the abscissas and the weights, respectively,
C     in the approximation of the b-integral.
C
C     xg1 and wg1 contain the abscissas and the weights, respectively,
C     in the approximation of the mu-integral.
C
C     xg2 contains the abscissas in the approximation of the p0-integral.
C
C     wg12 contains the product of the weights used in the approximation of
C     the mu-integral and p0-integrals.
C
C     intSplit contains p0,split(mu) for the abscissas used in the
C     approximation of the mu-integral.
C
C     ng1 is the number of points used in the approximation of
C     the mu-integral.
C
C     ng2 is the number of points used in the approximation of
C     the p0-integral.
C
C     ngauss is the number of points used in the approximation of
C     the b-integral.
C
      integer nmax
      parameter(nmax=1200)
      real*8 xg(nmax),wg(nmax)
```

```
      common /gauss1/ xg,wg
      integer ngauss
      common /gauss2/ ngauss

      real*8 xg1(nmax),xg2(nmax,2*nmax),wg1(nmax),
     *                  wg12(nmax,2*nmax),intSplit(nmax)
      common /pgauss1/ xg1,xg2,wg1,wg12,intSplit
      integer ng1,ng2
      common /pgauss2/ ng1,ng2
```